University of Pisa
Department of Computer Science

Ph.D. Thesis

INF/01

On Improving Stochastic Simulation for Systems Biology

Davide Cangelosi

Supervisor
Prof. Pierpaolo Degano

Supervisor
Dr. Roberto Marangoni

November 25, 2010

Abstract

Mathematical modeling and computer simulation are powerful approaches for understanding the complexity of biological systems. In particular, computer simulation represents a strong validation and fast hypothesis verification tool. In the course of the years, several successful attempts have been made to simulate complex biological processes like metabolic pathways, gene regulatory networks and cell signaling pathways. These processes are stochastic in nature, and furthermore they are characterized by multiple time scale evolutions and great variability in the population size of molecules. The most known method to capture random time evolutions of *well-stirred* chemical reacting systems is the Gillespie's *Stochastic Simulation Algorithm*. This Monte carlo method generates exact realizations of the state of the system by stochastically determining when a reaction will occurs and what reaction it will be. Most of the assumptions and hypothesis are clearly simplifications but in many cases this method have been proved useful to capture the randomness typical of realistic biological systems. Unfortunately, often the Gillespie's stochastic simulation method results slow in practice. This posed a great challenge and a motivation toward the development of new efficient methods able to simulate stochastic and multiscale biological systems. In this thesis we address the problems of simulating metabolic experiments and develop efficient simulation methods for well-stirred chemically reacting systems. We showed as a Systems Biology approach can provide a cheap, fast and powerful method for validating models proposed in literature. In the present case, we specified the model of SRI photocycle proposed by Hoff *et al.* [4] in a suitable developed simulator. This simulator was specifically designed to reproduce *in silico* wet-lab experiments performed on metabolic networks with several possible controls exerted on them by the operator. Thanks to this, we proved that the screened model is able to explain correctly many light responses but unfortunately it was unable to explain some critical experiments, due to some unresolvable time scale problems. This confirm that our simulator is useful to simulate metabolic experiments. Furthermore, it can be downloaded at the URL `http://sourceforge.net/projects/gillespie-qdc`. In order to accelerate the simulation of SSA we first proposed a data parallel implementation on *General Purpose Graphics Processing Units* of a revised version of the Gillespie's First Reaction Method [178]. The simulations performed on a `GeForce 8600M GS` Graphic Card with 16 stream processors showed that the parallel computations halves the

execution time, and this performance scales with the number of steps of the simulation. We also highlighted some specific problem of the programming environment to execute non trivial general purpose applications. Concluding we proved the extreme computational power of these low cost and widespread technologies, but the limitations emerged demonstrate that we are far from a general purpose application for GPU. In our investigation we also attempted to achieve higher simulation speed focusing on τ-leaping methods. We revealed that these methods implement a common basic algorithmic convention. This convention is the pre-computation of information necessary to estimate the size of the leap and the number of reactions that will fire on it. Often these pre-processing operations are used to avoid negative populations. The computational cost to perform these operations is often proportional to the size of the model (i.e. number of reactions). This means that larger models involve larger computational cost. The pre-processing operations result in very efficient simulation when the leap are long and many reactions can be fired. But at the contrary they represent a burden when leap are short and few reactions occur. So to efficiently deal with the latter cases we proposed a method that works differently respect to the trend. The SSALeaping method, SSAL for short, is a new method which lays in the middle between the direct method (DM) and a τ-leaping [182]. The SSALeaping method *adaptively* builds leaps and stepwise updates the system state. Differently from methods like the Modified τ-leaping (MTL) [39], SSAL neither shifts from τ-leaping to DM nor pre-selects the largest leap time consistent with the leap condition. Additionally whereas MTL prevents negative populations taking apart critical and non critical reactions, SSAL generates sequentially the reactions to fire verifying the leap condition after each reaction selection. We proved that a reaction overdraws one of its reactants if and only if the leap condition is violated. Therefore, this makes it impossible for the population to become negatives, because SSAL stops the leap generation in advance. To test the accuracy and the performance of our method we performed a large number of simulations upon realistic biological models. The tests aimed to span the number of reactions fired in a leap and the number of reactions of the system as much as possible. Sometimes orders of magnitude. Results showed that our method performs better than MTL for many of the tested cases, but not in all. Then to augment the number of models eligible to be simulated efficiently we exploiting the complementarity emerged between SSAL and MTL, and we proposed a new adaptive method, called *Adaptive Modified SSALeaping* (AMS). During the simulation, our method switches between SSALeaping (SSAL) and Modified τ-leaping, according to conditions on the number of reactions of the model and the predicted number of reactions firing in a leap. We were able to find both theoretically and experimentally how to estimate the number of reactions that will fire in a leap and the threshold that determines the switch from one method to the other and viceversa. Results obtained from realistic biological models showed that in practice AMS performs better than SSAL and MTL by augmenting the number of models eligible ro be simulated efficiently. In fact, the method selects correctly the best algorithm between SSAL and MTL according to the cases.

In this thesis we also investigated other new parallelization techniques. The parallelization of biological systems stimulated the interest of many researchers because the nature of these systems is parallel and sometimes distributed. However, the nature of the Gillespie's SSA is strictly sequential. We presented a novel exact formulation of SSA based on the idea of partitioning the volume. We proved the equivalence between our method and DM, and we have given a simple test to show its accuracy in practice. Then we proposed a variant of SSALeaping based on the partitioning of the volume, called Partitioned SSALeaping. The main feature we pointed out is that the dynamics of a system in a leap can be obtained by the composition of the dynamics processed by each sub-volume of the partition. This form of independency gives a different view with respect to existing methods. We only tested the method on a simple model, and we showed that the method accurately matched the results of DM, independently of the number of sub-volumes in the partition. This confirmed that the method works and that independency is effective. We have not already given parallel implementation of this method because this work is still in progress and much work has to be done. Nevertheless, the Partitioned SSAleaping is a promising approach for a future parallelization on multi core (e.g. GPU's) or in many core (e.g. cluster) technologies.

Acknowledgments

Firstly, I would like to thank people that are important in my life. In particular, my father Stefano and my mother Caterina that are two milestones of my existence. My brothers Francesco and Riccardo that always supported my decisions with smile and admiration. Catia and Marilisa that are always happy to know my novelties. My niece and nephews Stefano, Clara and Carla that remember me the enthusiasm to make everything. Alessia that fills my days even when she is not with me. Bonchi because when he says something my certainty totter and his paper [185] has been of inspiration. Lorenzo and Vito because they are my friends since I was young. My theater group Hengel, Sara, Igor, Lia, Dona because only you know most of my strengths and weaknesses. Sancasciani homes for sharing. Roberto because you always find time for me and Piepaolo because in spite of contrasts you let me to find my way. My friends and colleagues at the department for their happiness and for all time spent together. In particular, my room mates Fausto and Lorenzo and my friends Igor and Cristian who have special place in my heart. I am also very grateful to *Novartis Vaccines and Diagnostics Srl. member of Novartis* for financing my Ph.D. Many other people would to be cited here but I thank them all together.

To my family for their infinite trust and support
To my friends for telling me what I don't like to listen about me
To the theater for teaching me how I am and I could be
To the music for its power to awake feelings in me
To the research for giving me always a motivation to go beyond

Reviews and Comments to the Thesis

First Review. I found this thesis very enjoyable and rewarding to read. The work reported represents a serious and concerted attempt to improve upon the state-of-art in stochastic simulation of biochemical reactions. The author displays comprehensive knowledge of the current state-of-art and makes a number of significant and valuable contributions in proposing novel algorithms to improve upon those which are in widespread use today. The effectiveness of these novel algorithms is assessed thoroughly through a combination of mathematical derivation of their asymptotic time complexity and practical assessment of the their running complexity and, where appropriate for approximated accelerated methods, the accuracy of the results obtained. Up-to-date execution platforms are considered, such as general-purpose GPUs. Challenging physics scenarios are considered, such as use of non-well stirred models which are partitioned into sub-volumes. Taken in all, I believe that this represents a programme of work which is broad enough in scope, difficult enough in challenge, and rich in achievement to be worthy of the award of Ph.D.

Stephen Gilmore, University of Edinburgh, UK

Second Review. I enjoyed reading Davide Cangelosi‘s thesis titled *On Improving Stochastic Simulation for Systems Biology.* It has a good quality in writing. I like the literature review part very much. I think Davide did quite a thorough review for this area, which shows that he has had a lot of reading and understands very well the frontier line of the state-of-the-art research in this area. I also like the development of the software tool QDC and the idea the algorithm SSAL is based on. Based on the content in this thesis, I will recommend a pass for his PhD degree.

Yang Cao, Virginia Tech blacksburg VA, US

Contents

I Introduction **15**

1 Introduction on the mainstream research **17**

2 Other subjects **23**
2.1 Masking Patterns in Sequences: A New Class of Motif Discovery with Don't Cares . . . 23

3 Thesis plan **29**
3.1 Published (or pending) items about the work presented in this thesis 30

II Background **31**

4 Basics of Chemical Kinetics **33**
4.1 Continuous and Deterministic Formulation . . . 35
4.2 Continuous and Stochastic Formulation . . . 36
4.3 Gillespie's Discrete and Stochastic Formulation . . . 39

5 Simulation Algorithms and Improvements **43**
5.1 The original Gillespie's Exact Methods . . . 43
5.1.1 First Reaction Method . . . 43
5.1.2 Next Reaction Method . . . 44
5.1.3 Direct Method . . . 46
5.1.4 StochSim: an alternative to Gillespie's Exact Methods . . . 47
5.2 Exact Methods to Accelerate the SSA . . . 48
5.2.1 Exact Sequential Methods . . . 48
5.2.2 Exact Parallel and Distributed Methods . . . 49
5.3 Approximated Methods to Accelerate SSA . . . 50
5.3.1 τ-leaping methods. . . . 50
5.3.2 Hybrid methods. . . . 55
5.3.3 Multiscale methods. . . . 56
5.3.4 Accuracy Measurement . . . 59
5.4 Chemical Kinetics with Different Hypothesis . . . 61

5.4.1 Chemical kinetics with Spatiality 61
5.4.2 Chemical kinetics with Delayed Reactions 62

III Case Study for biochemical processes simulation 65

6 Modeling perceptive functions: the photoperception in *Halobacterium salinarum* 67
6.1 Overview 67
6.2 SRI's Photocycle 69
6.3 Phototaxis 70
6.3.1 Complex stimuli 72

7 QDC: Quick Direct Method Controlled 73
7.1 The approach: from biochemical systems to metabolic experiments . 74
7.2 Our tool: From input to output, SBML Import/Export and GUI . . . 75
7.2.1 QDC's syntax specification and input file structure 75
7.3 QDC's core 78
7.3.1 QDC's output 78
7.3.2 SBML Import/Export 79
7.3.3 QDC's GUI 79
7.4 Numerical experiments and Performance evaluation 79
7.4.1 Some simple example 79
7.4.2 A benchmark test 81
7.5 Conclusions 84

8 *In silico* testing of qualitative photoperception model 85
8.1 Data collection 85
8.2 The model 86
8.3 Results and Discussions 89
8.3.1 Dark (Absence of stimulation) 89
8.3.2 Long Lasting Orange 89
8.3.3 Blue Flash 92
8.3.4 Blue Flash on an orange background 92
8.3.5 Red and Blue Flash (white stimulus) 92
8.3.6 Sequence of orange pulses (Integration effect) 96
8.3.7 Orange step-down followed by a short orange pulse (Paradox effect) 96
8.3.8 Conclusions 99

IV Algorithmic Improvements to Stochastic Simulation 101

9 Optimized Parallel FRM on Graphics Processing Units 103
9.1 General-purpose GPU . . . 103
9.2 Sequential implementation . . . 107
9.3 Parallel implementation . . . 108
9.4 Result and discussion . . . 109
9.5 Conclusion . . . 111

10 SSALeaping: Efficient Leap Condition Based DM Variant 113
10.1 Stochastic Simulation Algorithm: DM . . . 114
10.2 Modified τ-leaping and Efficient τ-Formula . . . 115
10.3 Our Proposal: SSALeaping (SSAL) . . . 120
10.4 No Negative Populations . . . 120
10.5 Optimizations and Comparisons . . . 123
10.6 Numerical Experiments . . . 126
10.7 Decaying-Dimerizing Model . . . 126
10.8 Map Kinase Cascade Model . . . 128
10.9 LacZ/LacY Model . . . 128
10.10Performance Comparison . . . 132
10.11Conclusions . . . 132

11 Efficient Adaptive Tau-leaping Method 135
11.1 Adaptive Modified SSALeaping (AMS) . . . 136
11.1.1 Determining the threshold . . . 137
11.1.2 Estimating the occurrences in a leap . . . 137
11.1.3 Algorithmic Optimizations . . . 138
11.1.4 The Algorithm AMS . . . 139
11.2 Numerical Experiments . . . 142
11.2.1 Decaying-Dimerizing Model . . . 143
11.2.2 LacZ/LacY Model . . . 143
11.2.3 Map Kinase Cascade Model . . . 145
11.2.4 Performance and Accuracy Comparison . . . 145
11.3 Conclusions . . . 146

V Ongoing work 155

12 Ongoing Work 157
12.1 First Subvolume Direct Method . . . 157
12.1.1 Testing FSDM . . . 160
12.2 The Partitioning of the Volume . . . 162
12.2.1 Partitioned SSALeaping . . . 163

12.2.2 Testing PSSAL 163
12.3 Conclusions 166

VI Concluding Remarks and Future Work 169

Conclusions 171

Bibliography 175

Bibliography 175

Appendices 190

A QDC's Syntax 191

B Proof of Exactness of FSDM 197

List of Figures

6.1 The four-state model, from [7] 69
6.2 The seven-state model, modified from [4] 70

7.1 An example of QDC input file: blocks are separated by blank lines. . 80
7.2 QDC GUI input form. 80
7.3 QDC GUI: monitoring the running status. 81
7.4 QDC GUI output: plotting the time-course 82
7.5 An example of a stiff system input file. 83
7.6 Plotting the time-course of the chemical species concentrations for the stiff system. 83
7.7 Plotting the time-course of the reaction firing rates for the stiff system. 84

8.1 Our model for the photocycle 86
8.2 Signaling pathway from CheY-P to the flagellar motor 87
8.3 QDC input file for our model 88
8.4 Experimental response of a dark condition, from [12] 90
8.5 Simulation of a dark condition 90
8.6 Responses to orange light. Top: experimental behavior, modified from [8]; bottom: our model's response. 91
8.7 Our model's response to a blue light flash. 93
8.8 Blue flash on an orange background. Top: experimental response, from [2]; bottom: our model's response. 94
8.9 Top: red and blue flash response, modified from [5]; bottom: our result. 95
8.10 Top: stimulation with a sequence of orange pulses, modified from [3]; bottom: analogue stimulation in our model. 97
8.11 Top: *In vivo* response to an orange step-down followed by a short orange pulse, modified from [5]; bottom: the same stimulation in our model. 98

9.1 The GPU Devotes More Transistors to Data Processing 104
9.2 Thread Batching 105
9.3 Memory Model 106

9.4 Comparison of CPUTimes (in milliseconds) of the sequential and the parallel version of `FRM` algorithm considering different number of simulation steps.. 110
9.5 Speedups between sequential and parallel `FRM` considering different number of simulation steps. 111

10.1 Results comparison between SSAL and MTL for 1000 independent simulations for the Decaying-Dimerizing reactions. Accuracy have been taken for the values $X_0(10.0)$. 127
10.2 Results comparison between SSAL and MTL for 1000 independent simulations for the Map Kinase Cascade reactions. Accuracy have been taken for the values $X_2(0.1)$. 129
10.3 Results comparison between SSAL and MTL for 1000 independent simulations for the LacZ/LacY reactions. Accuracy have been taken for the values $X_{trrbslacy}(330)$. 131
10.4 Speed-up between SSAL and DM. 132
10.5 Speed-up between SSAL and MTL. 133

11.1 Average number of reactions fired in a leap after 1000 independent simulations varying ϵ from 0.003 to 0.06 and μ in the set $(4M/\log(M)$, $8M/\log(M)$, $16M/\log(M))$ for DD model. 144
11.2 CPUTime after 1000 independent simulations varying ϵ from 0.003 to 0.06 and μ in the set $(4M/\log(M)$, $8M/\log(M)$, $16M/\log(M))$ for DD model. 145
11.3 Number of leap executed with SSAL and MTL after 1000 independent simulations varying ϵ from 0.003 to 0.06 and $\mu = 16M/\log(M)$ for DD model. 146
11.4 Average number of reactions fired in a leap after 100 independent simulations varying ϵ from 0.003 to 0.06 and μ in the set $(4M/\log(M)$, $8M/\log(M)$, $16M/\log(M))$ for LacZ/LacY model in the time interval [300,330]. 147
11.5 CPUTime after 100 independent simulations varying ϵ from 0.003 to 0.06 and μ in the set $(4M/\log(M)$, $8M/\log(M)$, $16M/\log(M))$ for LacZ/LacY model in the time interval [300,330]. 148
11.6 Number of leap executed with SSAL and MTL after 100 independent simulations varying ϵ from 0.003 to 0.06 and $\mu = 16M/\log(M)$ for LacZ/LacY model in the time interval [300,330]. 148
11.7 Average number of reactions fired in a leap after 10 independent simulations varying ϵ from 0.003 to 0.06 and μ in the set $(4M/\log(M)$, $8M/\log(M)$, $16M/\log(M))$ for LacZ/LacY model in the time interval [600,700]. 149

11.8 CPUTime after 10 independent simulations varying ϵ from 0.003 to 0.06 and μ in the set $(4M/\log(M), 8M/\log(M), 16M/\log(M))$ for LacZ/LacY model in the time interval [600,700]. 149
11.9 Number of leap executed with SSAL and MTL after 10 independent simulations varying ϵ from 0.003 to 0.06 and $\mu = 16M/\log(M)$ for LacZ/LacY model in the time interval [600,700]. 150
11.10 Average number of reactions fired in a leap after 1000 independent simulations varying ϵ from 0.003 to 0.06 and μ in the set $(4M/\log(M), 8M/\log(M), 16M/\log(M))$ for MAPK model. 150
11.11 CPUTime after 1000 independent simulations varying ϵ from 0.003 to 0.06 and μ in the set $(4M/\log(M), 8M/\log(M), 16M/\log(M))$ for MAPK model. 151
11.12 Number of leap executed with SSAL and MTL after 1000 independent simulations varying ϵ from 0.003 to 0.06 and $\mu = 16M/\log(M)$ for MAPK model. 151
11.13 CPU Times after 1000 independent simulations for SSAL, MTL and AMS varying ϵ from 0.003 to 0.06 and considering $\mu = 16M/\log(M)$. 152
11.14 Accuracy for SSAL, MTL and AMS with $\mu = 16M/\log(M)$ for $\epsilon = 0.004$, $\epsilon = 0.01$ and $\epsilon = 0.06$ after 1000 independent simulations. . . . 153

12.1 Accuracy for FSDM for $\epsilon = 0.03$ after 1000 independent simulations. . 162
12.2 Accuracy for PSSAL for $\epsilon = 0.03$ after 1000 independent simulations and $N_{DIV} = 2$. 164
12.3 Accuracy for PSSAL for $\epsilon = 0.03$ after 1000 independent simulations and $N_{DIV} = 4$. 165
12.4 Accuracy for PSSAL for $\epsilon = 0.03$ after 1000 independent simulations and $N_{DIV} = 8$. 166
12.5 Average number of reactions fired in a leap changing N_{DIV} 166

List of Tables

4.1 Stochastic rate constant c_j and number of distinct R_j reactant combinations h_j for elementary reaction channels R_j. 38

7.1 Conversion table between stochastic and deterministic kinetics constants. N_A is the Avogadro's constant. 78

10.1 MTL statistics for the Decaying-Dimerizing reactions for 1000 independent run. 128

10.2 MTL statistics for the Map Kinase Cascade reactions for 1000 independent run. 130

10.3 MTL statistics for the LacZ/LacY reactions for 1000 independent run. 132

Part I

Introduction

Chapter 1

Introduction on the mainstream research

In the genomic era, it was argued that information about the functioning of living cell reside in the genome physically encoded in a DNA molecule. For years biologists collected data classifying the elements (e.g. genes and proteins) involved in the inner processes of living cells. But in the post-genomic era the overview changed. It emerged that the basic principles of the functioning of a biological system cannot be fully understood merely by drawing diagrams of the gene and protein interconnections. Therefore while an understanding of genes and proteins continues to be an important step, the focus of biology is shifted on understanding how the interactions between those elements give a specific cellular function, or a well defined phenotypic character or, at higher level, a physiological response. The shift from a static view to a dynamic one changed the way to think biological organisms leading toward a system level view. In order to understand biological organisms at system level, four key aspects need to be accomplished [100].

1. Understanding the structure of systems, including physical structures as well as the network of gene interactions and biochemical pathways.

2. Understanding of dynamics of systems, performing both qualitative and quantitative analysis as well as construction of theories/models with powerful prediction capability of the system's behavior.

3. Understanding of mechanisms for systematically control the state of a cell, to minimize malfunctions and devise effective therapies.

4. Devising of design methods to construct and modify biological systems with desired properties.

This level of systemic comprehension will provide substantial advantages and possibilities to biologists. For instance, it will help in new drug discovery, disease mechanism explanation and fast hypothesis verification [16]. For now, this task seems to be far because the counterpart of these great possibilities is the complexity characterizing living systems.

Real-life processes are inherently *multi-physic*s and *multi-scale*. From amino-acids to living organisms, nature builds systems that involve interactions amongst a wide range of physical phenomena operating at different spatial and temporal scales. Complex behaviors not necessarily emerge from a large number of interacting elements of the system. Sometimes it can just manifests itself from very few elements. In such a scenario a combination of experimental and computational approaches is expected to handle this complexity [13]. The dynamic and system level perspective is considered in the area of research called *Systems Biology*.

Systems Biology can be viewed as a *field of study*, in particular, the study of the interactions between the components of a biological system, that aims to understand how these interactions give rise to specific behaviors.

But also as a *paradigm*, usually defined in antithesis to the so-called reductionist paradigm. The distinction between the two paradigms is that the reductionist approach tries to understand simpler fundamental elements in order to understand the system as a whole. Instead, the holistic approach of Systems Biology regards the conviction that the properties of the whole cannot be explained by the properties of its parts, but they have to be studied directly by the whole. In this case people speak of *emergent properties*.

Systems Biology can also be viewed as a *working protocol to perform research*. A cycle composed of theory, analytic or computational modelling that propose specific testable hypotheses about a biological system, that validate them through experiments, and then refine the computational models or the theories using the knowledge acquired, restarting the cycle.

Another view can be the *socio-scientific phenomenon* defined by the strategy of pursuing integration of complex data deriving from diverse experimental sources using interdisciplinary tools.

Inside a cell, a large number of different entities (e.g. genes, proteins, inorganic chemical elements, ecc.) move in the space at a specific speed reacting selectively with other entities. The results of these reactions are new entities which often possess new reactive capabilities. In this way, the system evolves in time performing all activities necessary to the cell for living. The network of interactions among the entities forms a graph in which the nodes are the entities or species and the edges are the interactions or reactions among them. Reactions happens at specific rates depending on the number of species involved, on the number of molecules present in the space and on a specific basal rate (affinity) that depends on the number and type of species involved in that reaction. How molecules move, how fast they move, how they bump into each other, and how all that results in chemical reactions is competence of *Kinetics*. Computer simulation takes a description of the network, the law provided by the kinetics, an initial state and it mimics the temporal evolution of this system. This makes computer simulation well suited to do quantitative analysis of many biological systems.

Undoubtedly, *in silico* experiments can have an edge over conventional experimental biology in terms of cost, ease and speed. Having a complete description of a process, *in silico* experiments offers opportunities for unprecedented control over the system. Modeling can provide valuable insights into the working and general principles of organization of biological systems. It also can suggest novel experiments for testing hypotheses based on the modeling experiences [119]. However, *in silico* biology cannot be considered the substitute of the more conventional biology, that in any case remains fundamental.

A direct way to specify a network or model is to simply write down a list of reactions corresponding to the system of interest. However, in literature exists a strong research vein that focused on finding more expressive languages with respect to simply write down the list of reactions [171]. These formalisms view biological systems as distributed systems composed by a huge number of patterns that interact and compete, characterized by decentralized control and strong localization of interactions [15]. In the last decade a main research field has addressed the problems of concurrent and distributed computation, proposing abstract formal languages, as for instance process algebras, for the specification of concurrent computational processes. We will not give a complete list of such a formalisms because it is out of the scope of this thesis. For more details refer to [15, 14]. However, today no accordance exists on the best formalism to use.

For what regards the time evolution, historically for years biological systems have been viewed as essentially deterministic in character, with dynamics entirely predictable given sufficient knowledge of the state of the system (together with complete knowledge of the physics and chemistry of interacting bio-molecules). In recent years, has emerged the concern on the fact that reactions are more realistically modelled as discrete event, for which no prediction can be made about the exact time a reaction will occur. Many recent studies have confirmed the phenotypic variability of organisms to an inherent stochasticity that operates at a basal level of gene expression [119].

In such a scenario uncertainty deriving by reality must be treated probabilistically and consideration about statistical physics is necessary to uncover the precise nature of the stochastic process governing the system dynamics [171]. In many cases, when small numbers of molecules react in a stochastic manner, random fluctuations are observable and macroscopic concepts such as chemical concentration cannot be used to describe the stochastic dynamics [140]. One of the most used method to capture those randomness is through *stochastic simulation*. The *Stochastic Simulation Algorithm* (SSA) [75] proposed by Gillespie in the 70's is the most famous method to compute the random evolution of a system in a well-stirred chemical reacting environment. Time evolutions or realizations are generated moving the system forward in time by determining when will the next reaction occur and what reaction it will be. The Stochastic Simulation Algorithm was first realized giving two equivalent implementations known as the *Direct Method* (DM) [180] and the *First Reaction Method* (FRM) [75]. These original methods generate the next reaction to fire and its occurrence time in two probabilistically equivalent ways. They are also termed *exact* in the sense that the two algorithms sample the pairs according to theoretically founded and probabilistically correct procedures. However, the generation of each single reactive event makes SSA slow when it is used to simulate realistic biological systems.

This happens for three reasons mainly. The first is that the sequence of reactions and times represents one possible evolution of the system. The uncertainty about future evolutions requires to collect statistical information about the final states reached. So in general people that use SSA implementations need to perform an ensemble of independent simulations (e.g. 1000 runs). This represents an important distinction between the deterministic and the stochastic approach, and surely a computational cost. The second is that biological systems can evolve in different time scales. Sometimes when this happens a subset of the entire reactions are faster than the others. This would not be a problem in normal cases, however sometimes these fast reactions reach a stable state or equilibrium, and SSA spends very long time to simulate these fast reactions, but unfortunately this effort not correspond to a dynamical evolution of the system because the system is stable. This phenomenon is known with the name *Stiffness*, and it affects any approach used to simulate chemical reacting systems. The third represents the case in which the populations of some species are orders of magnitude larger than the others. In this cases, the reactions with these large population species are fast and occur many more times than the others. However, when populations are large the fluctuations typical of small numbers result less evident. So SSA spends long time to catch the dynamics of a system that would be better simulated with a less sensitive to fluctuations method.

Now, the reality of the simulation of biological systems can be very different and more complicated situations can happens. For example, at the beginning a system can exhibits stiffness, and then after some time it can require to track each single reaction, or it can happen that some species with large population amount reacts a lot.

During the years a lot of methods have been proposed to accelerate SSA implementations. Some of these methods maintain exactness making it faster the selection of the next reaction to fire and the consecutive updating of the state. Parallel methods that implement this form of acceleration decompose the volume into a number of sub-volumes. They divide the molecules into smaller independent populations assigning them to each sub-volume [154, 141]. The division leads these methods to suffer of some accuracy and sometimes efficiency problems. Recently, programmability improvements makes it possible the general purpose programming of the high performance computing technologies known as *Graphics Processing Units*(GPUs). In this thesis we attempt to overcome the above efficiency and accuracy problems proposing an exact parallel implementation of the First Reaction Method on NVIDIA GPU's.

Others methods sacrifice some exactness in order to achieve higher simulation speed [40, 145, 147, 136, 46, 44, 73, 131]. Among these the τ-leaping methods have been proved very efficient solutions. The basic idea of the original τ-leaping [73] is to divide the simulation into contiguous time intervals, called *leaps*. By enforcing a condition on the propensity functions associated to the reactions called *Leap Condition*, the original τ-leaping computes the reactions to fire in a leap in bulk. This simple but effective idea makes τ-leaping methods a natural connection between the stochastic and the deterministic regimes [76]. The advantage of this feature is that they deal efficiently systems with different population scales. Until now three fundamental points characterizes a τ-leaping method. These are: how it generates the largest leap time consistent with some definition of leap condition, the distribution it uses to sample the occurrences of the reactions and how it avoids negative populations. A common basic algorithmic convention of these methods is the pre-computation of information necessary for the leap. Often these pre-processings are selected in such a way that negative populations can be avoided. The algorithmic cost to perform these processing is often proportional to the size of the model. This means that larger models involve larger computational cost. This common convection results in very performant simulations when leap are long and many reactions can be fired. But unfortunately it can represent a burden when leap are short and few reactions occur. To deal efficiently these cases in this thesis we present a method called *SSALeaping*. This method lays in the middle between the direct method (DM) and a τ-leaping. It achieves good performance taking advantage from leap condition and the fast generation of the next reaction to fire. In a leap the SSALeaping method pays a computational cost proportional to the number of reactions fired. This makes SSALeaping faster respect to others τ-leaping methods provided that the number of reactions fired in a leap remains bounded. Our SSALeaping represents the connection between SSA and τ-leaping. So to augment the number of models eligible to be simulated efficiently in this thesis we present an adaptive method, called *AMS*, that exploit this complementarity. Our method switches between SSALeaping one of the most known and fast τ-leaping methods called Modified τ-leaping and viceversa. During the simulation the switch condition depends from the relation between the number of reactions of the model and the number of reactions expected to fire in that leap.

Furthermore, to handle efficiently the cases in which large number of reactions can be fired in a leap in this thesis we present an ongoing variant of SSALeaping based on the idea of partitioning the space into sub-volumes. This method is very different from other preceding τ-leaping or other simulation methods because it exploits a form of independency derived from the leap condition. This independency allows to separate the computation of the reactions to fire in a leap and to compose the results at the end of the leap. This operations can be done without loss of accuracy.

In this thesis we also address the problem of modeling and simulating the photo-motile responses of *Halobacterium salinarum*. In this work that involved the coordination and cooperation of a multisciplinary group, we use the simulation as fast hypothesis verification and validation tool. *Halobacterium salinarum* has been studied since thirty years, because of the fascinating and very complex motile responses it shows when exposed to different light stimuli. A very large collection of experimental data has been acquired, and some qualitative models have been proposed to explain the different response patterns shown after light stimulations. In the last few years, different simulators have been proposed that take into account the possibility to perform actions on the simulated biochemical systems, in order to have an *in silico* representation of metabolic experiments performed in wet-lab [133, 58, 143]. But in our opinion, they do not cover all the most frequent events that can take place in signaling pathways or metabolic experiments. So we developed a specific tool, that we called QDC (Quick Direct-method Controlled) designed to *in silico* reproduce wet-lab experiments.

Recently, more sensitivity about the applicability of the Gillespie's stochastic formulation of chemical kinetics deal scientists to propose new alternative formulations that relax some of the hypothesis made. Gillespie's formulation describes intracellular kinetics as a *well stirred* environment and it assumes that the reactions occur instantaneously. The well-stirred assumption holds in case of an equilibration of the reactants between all positions in the system volume occur on a much faster timescale than the chemical reactions time. Since diffusion of molecules in a living cell is considerably slower than in the test tube [64], the condition of spatial homogeneity and the well-stirred hypothesis are expected to be violated. Whereas the instantaneous assumption is true in many cases, it is also possible that some chemical reaction in living cells takes certain time to finish after they are initiated. Thus, the product of such reactions will emerge after certain delays [36]. This lead to take into account the possibility of delayed reactions.

Although the preceding alternative formulations are very interesting and certainly related to the topic of this thesis, they are not the focus of the present work.

Chapter 2

Other subjects

Even though the focus of this thesis is the Stochastic Simulation of chemically reacting systems, we want to briefly introduce our result in the area of biological sequence analysis.

2.1 Masking Patterns in Sequences: A New Class of Motif Discovery with Don't Cares

In this section, we describe the theoretical study of a new class of *motifs with don't cares*, motivated by sequence analysis in biological data and data mining on sequences [24]. Motifs are repeated patterns, where a pattern is an intermixed sequence of alphabet symbols (*solid* symbols) and special symbols $\circ$ (*don't care* symbols). The don't care symbol found in a position of the pattern specifies that the position may contain any alphabet symbol. For example, pattern `A∘T∘∘C` repeats twice in the input text sequence $T =$ `AAAATTACCCCATAGT` at positions 2 and 3 (starting from 0), and matches the two corresponding portions `AATTAC` and `ATTACC` of T.

Informally, motifs represent frequent patterns, where the latter ones repeat at least q times, for a user defined integer $q \geq 2$ called the *quorum*. Given an input text sequence T of length n, a quorum q, and a motif length L, we consider the problem of *motif discovery*: find the motifs of quorum q and length L in the text T. Each motif may have associated the list of the starting positions of its occurrences in the given sequence T. Unfortunately, due to the don't cares, the number of motifs can be exponentially large for increasing values of L. Potentially, there can be as many as $\Theta\big((|\Sigma|+1)^L\big)$ motifs, where Σ is the alphabet of the distinct symbols in the text T. Even though this number can be smaller for some particular instances, the known algorithms discovering these motifs still require, in the worst case, exponential time and space for increasing values of L. A lot of research has investigated these issues in order to mitigate the combinatorial explosion of motifs [65, 66, 120, 128, 134, 167].

Problem formulation and motivations We follow a new approach based on modeling motifs by using simple binary patterns, called *masks*, that implicitly represent *families of patterns* in T (instead of individual patterns). For example, mask `101001` represents both `A∘T∘∘C` and `T∘G∘∘A`: each `1` represents a solid symbol while each `0` represents a don't care symbol. A mask is a *motif* if at least one of its represented patterns occurs q or more times in the given sequence T.

As it should be clear from the above informal definition, we aim at describing interesting repetitions in a sequence, using a succinct description (mask) that gives rise to a smaller set of output motifs. Intuitively, consider some patterns that occur at least q times each and that also share the *same structure*, meant as a certain concatenation of solid and don't care symbols. Since they originate from the same mask, we take this mask as a motif. Moreover, any two patterns sharing the same structure but having a different number of occurrences in T (still at least q in number), which were previously considered as different motifs, are now giving rise to the same motif by our definition of mask. Since each mask can be seen as a binary string, we have potentially 2^L masks to examine instead of $(|\Sigma| + 1)^L$ frequent patterns with don't cares. In practice, the experimental tool `MaskMiner` in [24] found that the number of frequent patterns is actually close to the number of mask motifs, and so the $(|\Sigma| + 1)^L$ bound is overly pessimistic and we can use a lattice of 2^L masks as a better way to identify these patterns. Hence, our new class of motifs may summarize some regularities in the given sequence T, better than ever before.

We study the problem of detecting *maximal masks*, namely, the most specific ones (maximal number of `1`s) such that at least one of its represented patterns occurs q times or more. For example, given the text $T =$ `AAAATTACCCCATAGT`, fixing $L = 4$ and $q = 2$, we obtain the maximal masks `1110`, `0111`, and `1101`. Notice that `1110` and `0111` are equivalent since they originate the same patterns (three consecutive solid symbols) ignoring border effects, so we can treat them as the same mask. Therefore the patterns that are represented by the maximal masks are `AAA`, `CCC`, and `AA∘T`, and so the parameter L can equivalently be read as an upper bound on their length.

Specifically, we intend to solve the following *motif discovery* problem. We are given an input text sequence T over alphabet Σ, an integer length $L \geq 1$, and a quorum $q \geq 2$. We want to infer the set $\mathscr{M}$ of all *motifs* μ such that

1. μ is composed of L bits;

2. at least one of the patterns implicitly represented by μ occurs q or more times in T;

3. μ is *maximal*, namely, flipping any of its `0`s into a `1` violates condition 2 above.

It is worth noting that motif discovery has many applications in the investigation of properties of biological sequences. In such applications, it is a must to allow distinct occurrences of a motif to show some differences. In other words, we actually infer *approximated* motifs. Such approximation can be realized in several ways, according to the kind of application one has in mind. Motifs of limited length with don't cares can typically model biological object such as transcription factors binding sites, that are characterized by a short length, and a high conservation of their structure. Also, they present a high conservation of the contents in certain positions while for others it does not matter at all. The don't care symbols of our masks indeed aim at *masking* the latter, while the solid character should *unmask* the former.

Moreover, our masks could also be employed as building blocks for longer and flexible motifs, of different kind, allowing also indels. In recent years, there has been a growing interest in *seeds* for several applications (preprocessing filtration prior to a multiple alignment, approximate search task, data base search, BLAST like homology search, profile search, probe design) in bioinformatics ([68, 91, 102, 103, 101, 108, 158]). Among them, many have focused the attention on *gapped seeds*, or *spaced seeds* ([29, 32, 53, 60, 95, 157]). It turns out that gapped seeds can be found using the masks.

Finally, an application of finding motifs with don't cares could help to detect structural similarities, with a suitable input sequence. For example, when investigating the folding of a DNA sequence, it can be interesting to rewrite the sequence itself into the alphabet $\{\mathtt{w}, \mathtt{s}\}$ replacing each `A` and `T` with `w` (weak), and each `C` and `G` with `s` (strong). The motivation is that in the *base pairing* that assists in stabilizing the DNA structures, adenine (`A`) binds to thymine (`T`) via two hydrogen bonds, while cytosine (`C`) forms three hydrogen bonds with guanine (`G`). Hence, the latter bond is stronger than the former, and this has an influence on the actual structure of the molecule. Here, a motif on such sequence could represent a repeated structure, regardless of the actual DNA bases that form it. Further biological motivations can be found in [24].

Our results We show conceptually how to associate a pruned trie of height L with each mask μ. Since the text positions of the occurrences of the patterns implicitly represented by μ cannot overlap (while the patterns themselves can), we store the corresponding partition of the text positions into the trie, where the positions corresponding to the occurrences of the same pattern share a common leaf.

Our algorithm refers to the above pruned tries for the masks but it does not actually need to store them explicitly. Indeed, it extends the Karp-Miller-Rosenberg doubling scheme [93] and applies it to the masks, of length an increasing sequence of powers of 2 up to L. Our algorithm avoids to actually create the tries and just performs scanning and sorting of some suitable lists of consecutive pairs and triplets of integers. In this way, the access to memory is cache friendly.

However, the above method still generates the set $\mathscr{Q}$ of all the (maximal and not) masks having quorum q for the given sequence T, where $\mathscr{Q} \supseteq \mathscr{M}$. A postprocessing that filters from $\mathscr{Q}$ the masks that are not maximal, may increase the time complexity: precisely, it may take $\Theta(|\mathscr{Q}|^2 L)$ time in the worst case (e.g. [78]), yielding an additional cost of $\Omega(2^{2L} L)$ time.

We therefore introduce the crucial notion of *safe masks*, which includes the maximal masks as a special case. We show how to explore the lattice of 2^L masks of length L by examining only safe masks, so that maximal masks can be efficiently detected. In this way, we avoid the above postprocessing and obtain our final bound of $O(2^L n)$ time and space in the worst case, for discovering all the masks belonging to $\mathscr{M}$, which is our main result. Some tests in [24] show evidence of the advantages of this strategy also in practice.

In order to compare the time complexity of our proposed algorithm, consider the following scenario. After a preprocessing phase of the text T in polynomial time $O(n^c)$, for a constant $c \geq 1$, consider the following checking phase: for each of the $(|\Sigma| + 1)^L$ candidate patterns, verify if the given pattern has quorum and is maximal, taking just constant time (which is the best we can hope for, once a candidate pattern is given). An algorithm based on this ideal strategy would cost $O(n^c + (|\Sigma| + 1)^L)$ time. When the latter is compared to the $O(2^L n)$ time cost of our algorithm, we observe that $2^L n \leq n^c$ when $L \leq (c-1)\log_2 n$ and that $2^L n \leq (|\Sigma| + 1)^L$ when $L \geq \log_2 n/(\log_2(|\Sigma| + 1) - 1)$. Hence, our cost $O(2^L)$ is better than the ideal bound $O(n^c + (|\Sigma| + 1)^L)$ except for few degenerate cases (namely, when $c < 1 + (\log_2(|\Sigma| + 1) - 1)^{-1}$). In general, we can establish an upper bound $2^L n = O\big(n^{\Theta(1+1/\log_2|\Sigma|)} + \min\{2^L n, (|\Sigma| + 1)^L\}\big)$. In other terms, our algorithm performs better than virtually *constant-time enumerating and checking* all the potential $(|\Sigma| + 1)^L$ candidate patterns in T. In the above discussion for the complexity, we assume that the word size of w bits in the standard RAM is sufficiently large, so that $L = O(w)$. When L is much larger, the time complexity of our algorithm must be multiplied by a factor of $O(L/w)$.

Finally, given the scan-and-sort nature of our algorithm, we naturally obtain a cache friendly solution to our problem as a byproduct. To our knowledge, this is the first cache friendly solution for a motif discovery problem, which is useful for long input sequence(s). Indeed, our algorithm works also in the *ideal cache model*, introduced by Frigo et al. [71] to generalize the two-level memory model of Aggarwal and Vitter [17] and to deal with such a situation, where M is the size of the fast memory, and B is the size of the block in each transfer between fast and slow memories. The goal is to minimize the number of block transfers. For example, scanning n consecutive elements has a complexity of $\Theta(n/B)$ block transfers while the optimal complexity of sorting is $sort(n) = \Theta\big(\frac{n}{B}\log_{M/B}\frac{n}{B}\big)$ block transfers [31, 69, 71].

Employing the simple scan and the cache-oblivious sorting in our algorithm, we do not need to further orchestrate their memory accesses. Using this model, we can easily obtain a complexity of $O(2^L\, sort(n))$ block transfers for finding the masks in $\mathscr{M}$. Also, we think that our algorithm can run in a distributed setting, such as a cluster of computers, using distributed sorting.

Related problems and state of the art We are not aware of previous works introducing our class of motifs. Hence, we relate our results in motif discovery to those of mining frequent itemsets, where more sophisticated techniques have been found over the years. The notion of masks comes naturally into play when performing data mining for frequent itemsets, where the "apriori" algorithm is intensively employed [84]. Here, a set of L items is given, and each transaction (basket) corresponds to a subset of these items, which can be represented as a binary sequence in which the ith symbol is `1` if and only if the ith item is chosen for the basket. A set of baskets can be therefore represented as a set of masks in our terminology. For the lattice of all possible 2^L masks, all possible itemsets should be examined. Note that, instead, our definition of masks has the goal of condensing patterns that have the same sequence of solid and don't care symbols. Moreover, our traversal of the lattice is different from the apriori algorithm, since we start from the top and generate candidates in a different way, namely, using safe masks.

As far as we know, the "dualize and advance" algorithm [80, 81] is the best theoretical approach that can be obtained in terms of running time. It sets up an interesting connection between mining itemsets in the lattice of 2^L masks and finding hypergraph traversals [25]. In our terminology, suppose to have incrementally found some of the maximal masks, say $\mu_1, \mu_2, \ldots, \mu_k$. We build the corresponding hypergraph as follows: there are L nodes numbered from 1 to L, and there is one hyperedge per mask, where the jth bit in the mask is `0` if and only if the node j is incident to the corresponding hyperedge ($1 \leq j \leq L$). In general, the ith hyperedge connects the nodes that correspond to the `0`s in the ith mask μ_i ($1 \leq i \leq k$). In order to find additional maximal masks (and hence add hyperedges), it suffices to find all the hypergraph traversals as starting points for upward paths in the lattice, where each traversal is a minimal hitting set for the current set of k hyperedges [25].

The problem of finding hypergraph traversals is intimately related to the dualization of monotone Boolean functions [62]. The known algorithms required $O(2^L)$ time in the worst case [25, 94] until the seminal result in [70, 96] showing a subexponential bound proportional to $t(k) = k^{O(\log k)}$ time, when the number of hyperedges k is $o(2^L)$. This algorithm is plugged into the scheme of the "dualize and advance" algorithm, giving a bound of $O\left(n \times t(|\mathscr{M}| + |Bd^-(\mathscr{M})|)\right)$ as shown in [80], where we include the cost $O(n)$ of verifying the quorum, and $Bd^-(\mathscr{M})$ is a set of non-maximal masks that are "close" inside the lattice to the ones in $\mathscr{M}$. While $|\mathscr{M}|$ can be subexponential, there are cases in which $|\mathscr{M}| + |Bd^-(\mathscr{M})| = \Theta(2^L)$ [80], and so the final bound can be $\Omega(2^{L^2} n)$.

Surprisingly, this and other approaches based on hypergraph traversals, which are the state of the art theoretically, are slower than our solution in the worst case. We also run some experiments in [24] and found that our solution is faster in practice, where we accounted for the number of masks we queried for checking their quorum. Indeed, the dualize and advance method needs to query many more masks than our safe masks, thus suggesting that the latter notion is crucial to our algorithms.

Chapter 3

Thesis plan

This thesis is organized as follows.

The *Section entitled Background* gives the background and the state of the art in the area of stochastic simulation methods. In particular, a slightly description of the main formulations of chemical kinetics is provided in Chap. 4. A very detailed collection of exact and approximated methods have been surveyed in Chap. 5. In this listing we point particular attention on τ-leaping methods that are one of the focus of this thesis. This entire section is the result of my personal investigation in the area.

The *Section entitled Motivating Case Study* summarizes the contributions of our multisciplinary group, called *BioLab*. Details about the photo-perception of *Halobacterium Salinarum* are given in Chap. 6. Whereas the main features of our new simulator are presented in Chap. 7 and the results of our systems biology approach to the photo-motile responses case study are presented and discussed in Chap. 8. In particular, my main contributions in the QDC Tool and *Halobacterium Salinarum* project have been the following. The formal definition of the QDC input file format, the testing and the development of the tool and its examples. The surveying of the tools available and the performance comparison. But also the Halobacterium models development and hypothesis testing, group discussions and meetings since the early stages of the work. Writing and revision of the papers.

The *Section entitled Algorithmic Improvements to Stochastic Simulation* collects our proposed exact and approximated simulation methods. In Chap. 9 we describe practical issues arising from a parallel implementation of the First Reaction Method on 'Graphics Processing Units' (GPUs). In particular, my main contributions are the following. The sequential implementation, test cases, random number generation, development supervision and optimization. Writing and revision of the paper. In Chap. 10 we present *SSAL* a new variant of the stochastic simulation algorithm which lays in the middle between the Gillespie's Direct Method and a τ-leaping. We present AMS a new adaptive method in Chap. 11. Both SSAL and AMS are the result of my personal investigation in the area.

In the *Section entitled ongoing work* we introduce one of our ongoing simulation methods based of the partitioning of the volume. Its description and some numerical tests are given in Chap. 12. This on going method is the result of my personal investigation in the area.

Finally, in the *Part Concluding Remarks and Future Works* some concluding remarks and research perspectives are discussed. In Appendix A we present a formal syntax of our simulator, and in Appendix B provides the proof of exactness of one of the methods based on the partitioning of the volume. This part is the result of my personal investigation in the area.

3.1 Published (or pending) items about the work presented in this thesis

This thesis is also based on the following published and submitted results.

1. Chap. 2 has partly been published in [1].
2. Chap. 9 has partly been published in [178].
3. Chap. 10 has partly been published in [182].
4. Chap. 11 has partly been submitted to the 8th Conference on Computational Methods in Systems Biology, (CMSB 2010) in Trento (Italy).
5. Chap. 7 has partly been submitted to the Journal *IET Systems Biology*.
6. Chap. 6 and Chap. 8 have partly been published in Photoperception in *Halobacterium salinarium*: a systems biology approach. vol. unico, p. 1, *Poster* at the European Conference on Computational Biology (ECCB 2008) and Simulating signaling pathways: the motile photoresponse of H. salinarum as a case study, *Poster* associated to oral communication, BITS '09 Sixth Annual Meeting of the Bioinformatics Italian Society March 18 - 20, 2009, Genoa, Italy.

Finally, part of Chap. 12, Chap. 6 and Chap. 8 still have to be published as papers.

Part II

Background

Chapter 4

Basics of Chemical Kinetics

A cell has a large number of functionally diverse, and frequently multi-functional, sets of molecules interacting selectively and nonlinearly[1] to produce coherent behaviors. Dense networks of interacting macromolecules (genes, mRNAs, proteins), or *pathways*, control any cellular process [159]. Generally speaking, molecules move inside a cell with a certain speed and they bump into others molecules. By collision theory we know that a reaction occurs only when reactants collide, bumping into specific domains of their structure at a certain speed. How molecules move, how fast they move, how they bump into each other, and how all that results in chemical reactions is competence of *Kinetics*. Biochemists use an understanding of kinetics to figure out reaction *rates*. The rate of a chemical or biochemical reaction is just a measure of how the concentration of the involved substances changes with time. Due to the complexity of the pathway and molecular interactions it is almost impossible to intuitively predict the behavior of cellular networks. Mathematical modeling and computer simulation techniques have proved useful for understanding the topology and dynamics of such networks. In particular, the main ability of computer simulation is to mimic the temporal evolution of a set of elements that react according to the rules of kinetics to produce certain dynamics. This makes computer simulation well suited to do quantitative analysis of many biological systems. In silico biology has an edge over conventional experimental biology in terms of cost, ease and speed. Also experiments that are infeasible *in vivo* can be conducted *in silico*, e.g. it is possible to knock out many vital genes from the cells and monitor their individual and collective impact on cellular metabolism. Evidently such experiments cannot be done *in vivo* because the cell may not survive. The development of predictive *in silico* models offers opportunities for unprecedented control over the system. Modeling can provide valuable insights into the working and general principles of organization of biological systems. Also it can suggest novel experiments for testing hypotheses, based on the modeling experiences [119].

The modelling of chemical reactions using deterministic rate laws has been proved

[1]Usually the rate of production of a fired reaction cannot be expressed as a linear combination of the reactants concentration.

successful in both chemistry [150] and biochemistry [88] for many years. This deterministic approach has at its core the *law of mass action*, an empirical law giving a simple relation between reaction rates and molecular component concentrations. Given knowledge of initial molecular concentrations, law of mass action provides a complete picture of the component concentrations at all future time points [67].
The law of mass action considers chemical reactions to be macroscopic under convective or diffusive stirring, continuous and deterministic [57]. These are evident simplifications, as it is well understood that chemical reactions involve discrete, random collisions between individual molecules. As we consider smaller and smaller systems, the validity of a continuous approach becomes ever more tenuous. As such, the adequacy of the law of mass action has been questioned for describing intracellular reactions [148, 82]. Arguments for the application of stochastic models for chemical reactions come from at least three directions [166], since the models take into consideration the discrete character of the quantity of components and the inherently random character of the phenomena; they are in accordance (more or less) with the theories of thermodynamics and stochastic processes; and they are appropriate to describe instability phenomena. At the molecular level, random fluctuations are unavoidable. The effect of these fluctuations becomes more evident as we consider systems with smaller populations of molecules. This typically occurs in the regulation of gene expression where transcription factors interact with DNA binding sites in the genes regulatory sequences. Additionally, it has been proven that small numbers of expressed RNAs can be significant for the regulation of downstream pathways [117]. Thus, there are evidently a number of important biological environments where only small numbers of molecules are present in the reaction volume, for which, it is argued, stochastic modelling approaches are required [55]. There is also growing evidence of the importance for reaction kinetics of the structural organization of the intracellular environment, which is far from the homogeneous, well mixed solution typical of *in vitro* conditions (see [151] and references therein). Cellular environments are highly compartmented and structured throughout the reaction volume. A high degree of molecular crowding as well as the presence of endogenous obstacles in cellular media have important consequences in the thermodynamics of the cell [121] and strongly affect diffusion processes [112]. The viscosity of the mitochondrion is 2537 times higher than that of a typical *in vitro* experimental buffer [149]. Diffusion of macromolecules in the cytoplasm can be 520 times lower than in saline solutions [168]. Furthermore, many reactions occur on two-dimensional membranes or one-dimensional channels. These structural considerations mean that we must be careful when considering how well mixed a chemical system is. Apart this there is a strong research vein that believes that the discrete and stochastic nature of many biological process can only be captured by a stochastic approach.

The stochastic approach uses the inherent random nature of microscopic molecular collisions to build a probabilistic model of the reaction kinetics. This approach is thus inherently suited to the small, heterogenous environments typical of *in vivo* conditions [82]. In the next sections we introduce the basic principles of the Continuous and Deterministic, the Continuous and Stochastic and the Discrete and Stochastic formulation of chemical kinetics.

4.1 Continuous and Deterministic Formulation

The deterministic approach regards the time evolution as a continuous and wholly predictable process governed by a set of coupled, first order, ordinary differential equations, called *Reaction Rate Equations* (RRE). In general, the deterministic approach assumes that for large population of molecules the stoichiometric changes induced to a population by one reaction are small enough to assume that the overall changes to the population are continuous. Then for a large population of reactants the system behavior is well approximated by the average behavior. In other words, the fluctuations in the molecular populations due to the casual occurrence of the reactions introduces so negligible changes in the macroscopic trend of the concentrations that the mean value well describes the system evolution. Furthermore, the system is in thermodynamic equilibrium, and no temperature and volume changes occur.

Now given a set of elementary reactions R_j and set of species S_i RRE is a set of ODEs, one for each species.

The *Ordinary Differential Equation* (ODE) for S_i is obtained from the reactions in which S_i appear as reactant or product and from the rules of mass action kinetics. The *Mass action kinetics* states that the rate of any given elementary reaction is proportional to the product of the concentrations of the species reacting (reactants) in the reaction.

To show an example of how to set up RRE we consider system in Eq. 4.1. In this Michaelis-Menten model there is an initial bimolecular reaction between the enzyme E and substrate S to form the enzymesubstrate complex C. Although the enzymatic mechanism for the unimolecular reaction $C \xrightarrow{k_2} E + P$ can be quite complex, there is typically one rate-determining enzymatic step that allows this reaction to be modelled as a single catalytic step with an apparent unimolecular rate constant k_2.

$$S + E \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} C \xrightarrow{k_2} E + P \tag{4.1}$$

For example, the rate of production of the complex C, denoted with $\frac{d[C_+]}{dt}$, would be

$$\frac{d[C_+]}{dt} = k_1 SE$$

and the rate of destruction of the complex C, denoted with $\frac{d[C_-]}{dt}$, would be

$$\frac{d[C_-]}{dt} = k_{-1}C + k_2 C.$$

Combining these terms it results an ODE for the following rate of change of concentration of C

$$\frac{d[C]}{dt} = \frac{d[C_+]}{dt} - \frac{d[C_-]}{dt} = k_1SE - (k_{-1} + k_2)C \tag{4.2}$$

As we showed in the example above RRE are based on concentrations, while traditionally the discrete models are based on number of molecules. Exists a specific connection between concentrations and populations that can be summarized as follows.

The *molar concentration* of a given species S_i is denoted with $[S_i]$. It is defined as the number of moles of a solute dissolved in a liter of solution. Usually, molar concentration is given as mole per liter:

$$1\frac{mol}{L} \equiv 1M$$

For the International System of Units (SI) a mole is the base unit measurement of the amount of substance. One mole represents $6.02214179 * 10^{23}$ molecules and it was named the Avogadro Number in honor of the father of stoichiometry. Even though the Avogradro's number does not actually determine the exact number of molecules it represents the best approximation obtained with the best measurement method known. Here, we consider

$$N_A = 6.02214179 * 10^{23} mol^{-1}.$$

To convert a concentration into a number of molecules we need to consider the volume V, for instance expressed in liter (L), and N_A. Therefore, given the molar concentrations $[S]$ for a specie S expressed in mol per liter (M), and given a volume V the corresponding number of molecules $\sharp S$ follows the following expression

$$[S] * V = S \text{ and } \sharp S = S * N_A$$

where S is the number of moles in $[S]$.

4.2 Continuous and Stochastic Formulation

The validity of the assumptions made in the continuous and deterministic formulation becomes strained as we examine small-scale cellular reaction environments with limited reactant populations. Instead of dealing with only one possible evolution in time, as is the case for ODEs, the stochastic formulation takes into account the indeterminacy about future evolutions. This uncertainty is described by probability distributions. This means that even though the initial state is known, some evolutions can be more probable than others. So executing a bunch of time evolutions, a subset of the possible final states will be reached more frequently than others.

The stochastic formulation of chemical kinetics considers a system of N molecular species $\{S_1, \cdots, S_N\}$ interacting through M chemical reactions $\{R_1, \cdots, R_M\}$. It assumes that the system is *well stirred*, in a constant volume $\mathtt{V}$ and in thermal (but not chemical) equilibrium. Well stirred means that the overwhelming majority of molecular collisions that take place in the system are elastic (nonreactive), and the net effect of these elastic collisions is twofold [76]. First, the positions of the molecules become uniformly randomized throughout V. Second, the velocities of the molecules become thermally randomized to the Maxwell-Boltzmann distribution. It corresponds to the most probable speed distribution in a system consisting of a large number of non-interacting particles in which quantum effects are negligible. To the extent that this happens, the nonreactive molecular collisions can be ignored and only events that change the populations of the chemical species are considered. In this case, the system state is described by the multivariate variable $X(t) = \{X_1(t), \cdots, X_N(t)\}$, where $X_i(t)$ is the number of molecules of species S_i in the system at time t. The evolution in time of the system state is of course consequence of chemical reactions, and the reactions in the stochastic formulation of chemical kinetics approach are viewed as *distinct*, essentially *instantaneous* physical events of two elemental types. Unimolecular, occurring as a result of processes internal to a single molecule, and bimolecular, occurring as a result of a collision between two molecules. Fundamental to the principal of stochastic modelling is the idea that reactions are essentially instantaneous elementary random events. In this scenario, each reaction R_j can be characterized by a *propensity function* a_j and a *state change vector* $\nu_j \equiv (\nu_{1j}, \cdots, \nu_{Nj})$. Let $X(t) = x$, the quantity $a_j(x)dt$ gives the probability that one reaction R_j will occur in the next infinitesimal time interval $[t, t + dt)$, while ν_{ij} gives the change in the S_i molecular population induced by the reaction R_j.

To estimate the probability of a reaction to occur the stochastic formulation states the existence of a constant c_j which depends only on the physical properties of the molecules involved and the temperature. The stochastic reaction constant c_j is also termed the *fundamental hypothesis* of the stochastic formulation of chemical kinetics. Multiplying the probability $c_j dt$ for the total number of distinct combinations of reactants of R_j in V at time t, the result is

$$c_j h_j dt = a_j dt$$

that is equal to

$$P_j(dt) = a_j dt \tag{4.3}$$

where $P_j(dt)$ gives the probability that R_j will occur in V in [t, t+dt), provided that the system is in state X at time t.

Gillespie in [180] has given physical rationale of the propensity for unimolecular and bimolecular reactions that can be briefly summarized as follows.

If R_j is a unimolecular reaction of the form $S_1 \xrightarrow{c_j} Product(s)$ the underlying physics, dictates the existence of some constant c_j, such that $c_j dt$ gives the probability that any particular S_1 molecule will so react in the next infinitesimal time dt. It then follows from the laws of probability that if there are currently $\mathbf{x}_1$ molecules on the specie S_1 in the system, the probability that some one of them will undergo the R_j reaction in the next dt is $\mathbf{x}_1 c_j dt$. Thus the propensity function is $a_j(\mathbf{x}) = c_j \mathbf{x}_1$.

If R_j is a bimolecular reaction of the form $S_1 + S_2 \xrightarrow{c_j} Product(s)$, kinetic theory arguments and the well-stirred condition together imply the existence of a constant c_j, such that $c_j dt$ gives the probability that a randomly chosen pair of S_1 and S_2 molecules will react according to R_j in the next infinitesimal time dt. The probability that some one of the $\mathbf{x}_1 \mathbf{x}_2$ $S_1 - S_2$ pairs inside V will react according to R_j in the next dt is therefore $\mathbf{x}_1 \mathbf{x}_2 c_j dt$. In this case, the propensity function is $a_j(\mathbf{x}) = c_j \mathbf{x}_1 \mathbf{x}_2$. If instead the bimolecular reaction had been $S_1 + S_1 \xrightarrow{c_j} Product(s)$, the number of distinct S_1 molecular pairs are $\mathbf{x}_1(\mathbf{x}_1 - 1)/2$, and so the propensity function results $a_j(\mathbf{x}) = c_j \frac{1}{2} \mathbf{x}_1(\mathbf{x}_1 - 1)$.

Intuition suggests that the stochastic reaction constant c_j, should be closely related to the more familiar reaction-rate constant k_j, which forms the basis for the deterministic approach to chemical kinetics. Table 7.1 summarizes the mathematical relation between the stochastic coefficient and the deterministic reaction constants for each basic reaction type R_j. Note that the relationship between the stochastic

R_j	Units of k_j	c_j	h_j
$\emptyset \xrightarrow{k} \dots$	$M sec^{-1}$	kN_AV	1
$S_1 \xrightarrow{k} \dots$	sec^{-1}	k	X_{S_1}
$S_1 + S_2 \xrightarrow{k} \dots$	$M^{-1}sec^{-1}$	$kN_A^{-1}V^{-1}$	$X_{S_1} \cdot X_{S_2}$
$2S_1 \xrightarrow{k} \dots$	$M^{-1}sec^{-1}$	$2!kN_A^{-1}V^{-1}$	$\binom{X_{S_1}}{2}$

Table 4.1: Stochastic rate constant c_j and number of distinct R_j reactant combinations h_j for elementary reaction channels R_j.

rate constant c_j and the reaction rate k is only a constant factor. However, the conceptual difference is rather more complicated. The stochastic rate constant c_j, multiplied for h_j, is a propensity and thus referring to a stochastic model of a population of molecules. In contrast, the rate constants k_j are indeed rates in the sense of a velocity. The fact that the c_j, and hence any stochastic simulation, is dependent on the knowing the rate constants of the mass action model is no coincidence. It has not been feasible to derive an expression for c_j from physical principles without knowledge of either the rate constants or the probabilities that a colliding set of reactants of R_j will chemically react.

Now, in the stochastic formulation the probabilistic nature of the problem precludes making an exact prediction of $X(t)$, so what one might hope to infer is the probability that at time t in V there will be x_1 molecules of specie S_1, x_2 molecules of specie S_2, until x_N molecules of specie S_N starting from $X(t_0) = x_0$. This probability called *grand probability function* (GPF,) and denoted with $P(x,t|x_0,t_0)$, evolves according to a time-evolution equation known as *Chemical Master Equation* (*CME*). In probability theory, Eq. 4.4 identifies a *continuous-time Markov process*, that is, a mathematical model for the random evolution of a system for which, at any given moment, a given future state depends only on its present state, and not on any past states.

$$\frac{\partial P(x,t|x_0,t_0)}{\partial t} = \sum_{j=1}^{M} \left[a_j(x-\nu_j)P(x-\nu_j,t|x_0,t_0) - a_j(x)P(x,t|x_0,t_0)\right], \tag{4.4}$$

The preceding CME was obtained from laws of probability and the definition of propensity function. The first term on the right-hand side in Eq. 4.4 describes the probability from any other state to reach $X(t) = x$ in one step, while the second term describes probability to be just in $X(t) = x$ and change away. In principle, the CME completely determines the function $P(x,t|x_0,t_0)$. Unfortunately the major difficulty with the CME is that each chemical species in the model adds one dimension to the problem, and the computational time to obtain any numerical solution to the CME growths exponentially with the number of reacting species in the system. In Mathematics this phenomenon is known as *curse of dimensionality.* The main difficulty when the curse of dimensionality happens is that the analytical solution of the CME becomes intractable, in particular, if many reactions are bimolecular [76]. Supposing to be able to find an analytical solution for a CME, this would be a *Probability Density Function* (PDF) of $X(t)$ with a specific mean and variance. If we would compare that solution with the solution of a RRE for the same system, we would observe that in general we obtain different results for the means. This happens in presence of reactions with two reactant species because one considers propensity as function of the volume V whereas the other never.

4.3 Gillespie's Discrete and Stochastic Formulation

Because the CME can rarely be solved for the PDF of $X(t)$, Gillespie proposed a rigorous Monte Carlo procedure, known as *Stochastic Simulation Algorithm* (SSA), to generate numerical realizations of $X(t)$. To simulate numerically the time evolution this procedure moves the system forward in time by answering two questions: *when will the next reaction occur and what reaction will it be*? The main theoretical construct of the SSA is the *reaction probability density function* $P(\tau, j)$ defined as follows.

$$P(\tau,j)d\tau \equiv \begin{array}{c}\text{probability at time t that the next reaction}\\ \text{in V will occur in the infinitesimal time interval}\\ (t+\tau, t+\tau+d\tau)\text{, and will be reaction } R_j.\end{array} \tag{4.5}$$

In the terminology of probability theory, $P(\tau, j)$ is a joint probability density function (JPF) on the space of continuous variable τ $(0 \leq \tau < \infty)$ and the discrete variable j $(j = 1, 2, \cdots, M)$. However, SSA provides only the steps composing the procedure, and two logically equivalent implementations of SSA called *First Reaction Method* (FRM) [75] and *Direct method* (DM) [180] have been proposed. We will introduce them in detail in the next chapter.

The SSA and CME are *logically equivalent*. This means that they are rigorous consequences of the same premises [77]. In addition, to the limit of an infinite number of independents SSA realizations, the PDF $P(x, t|x_0, t_0)$ computable on the final states reached by the SSA realizations matches exactly the PDF $P(x, t|x_0, t_0)$ obtained solving analytically CME. This introduce an important point. A single realization gets no statistical picture of the temporal evolution of $X(t)$. A complete picture can be found only carrying out an infinite independent realizations. In practice, *a set of independent* realizations must be carried out to have an acceptable statistical picture.

The Stochastic Simulation algorithm has been applied to simulate different realistic biological models. For example, Chiarugi et al. simulated a Virtual Cell [52], Bracciali et al. modelled and simulated synaptic plasticity [27], Arkin [20] showed how stochastic variations in the concentrations of some regulatory protein can produce probabilistic pathway selection.

One important characteristic of SSA is the *exactness*. Often it was not clear what exact refers to. Gillespie states that SSA produces exact "realizations" of the jump Markov process $X(t)$ [77]. Sometimes exact refers to the fact that τ in SSA is not a finite approximation to some infinitesimal dt [76]. Another definition refers to the fact that under the hypothesis of the stochastic formulation of chemical kinetics, the systematic Monte Carlo generation of (τ, j) realizes *one possible* evolution of $X(t)$, probabilistically correct and theoretically founded. The preceding assertions are all undoubtedly true. The key point is that the exactness depends by the validity of its fundamental hypothesis of SSA to model realistic systems [124]. In fact, SSA is often applied to systems that not necessary respect the Gillespie's hypothesis. For instance, it is quite common to use SSA to simulate Biochemical systems. Biochemical systems can be seen as particular instances of chemical systems where reacting molecules are heavy biological compounds such as proteins and nucleic acids. However, these types of compounds are far from gaseous conditions as theoretically assumed by Gillespie's. Normally, in the small volume of a living cell no temperature gradients can be reasonably assumed, which entitles us considering the hypothesis of thermal equilibrium applicable. Similarly, the diffusion processes in a cell are also quite efficient. Even though each type of cell uses indeed various mechanisms to regulate the concentration of molecules in different areas, it is widely accepted to consider that in small volumes homogeneity is assured [124]. This observation would lead us to consider well-stirred hypothesis valid for most biochemical systems considered in limited volume areas.

Instead, more discussions are required to the hypothesis that the reactions are instantaneous. In a bimolecular reaction, the time of the reaction to occur can be considered negligible with respect to the collision time only if it does not involves complex transformations, such allosteric changes, of the two binding molecules. In biochemistry the reactants may be heavy structured molecules whose binding may be just the first step of a conformational rearrangement. In fact, the situation mentioned above is a particular example of the difficulty that can be encountered in describing a biochemical system in terms of elementary reactions. This is the case for the well-known Michaelis-Menten abstraction for the reactions in Eq. 4.1. Obviously, the hypothesis of having only elementary reactions is not always applicable, and this introduce approximations. However, the fundamental hypothesis can be considered valid using the Michaelis-Menten abstraction if the speeds of the binding and unbinding of the enzyme is much higher than the one of the catalysis reaction. Other cases of approximations can be found in [124].

In conclusion, as stated by Mura there is no general theory to predict a priori the effect that different model choices will have on output results. So the results obtained through the SSA must be validated with the observed behavior of the real system, if possible.

Chapter 5

Simulation Algorithms and Improvements

5.1 The original Gillespie's Exact Methods

As anticipated in the previous chapter, Gillespie proposed two equivalent procedures to perform exact realizations. This procedures are: the *First Reaction Method*(FRM) [75] and the *Direct method* (DM) [180]. Although these are famous another famous implementation is *Next Reaction Method*(NRM), proposed by Gibson et Bruck [72]. Below we briefly introduce them.

5.1.1 First Reaction Method

Every implementation of SSA are required to define the probability at time t that the next reaction in the considered volume V will occur in the infinitesimal time interval $(t + \tau, t + \tau + d\tau)$, and this reaction will be R_j. First Reaction Method defines it as follows.

$$P^{FRM}(\tau, j \mid \mathbf{x}(t))d\tau = a_j e^{-a_j \tau} d\tau. \tag{5.1}$$

The generation of a random pair (τ, j) happens by sampling it according to the PDF in Eq. 5.1. To do that for all M reactions, FRM generates a "tentative reaction time" τ_j according to the following formula

$$\tau_j = \frac{-\ln(r_j)}{a_j}, \tag{5.2}$$

and then it selects as next firing reaction that one which occurs first, i.e. the method takes

$$\tau = \text{ smallest } \tau_j \text{ for all } j = 1, \cdots, M \tag{5.3}$$

and

$$j = \text{ reaction index j for which } \tau_j \text{ is the smallest }. \tag{5.4}$$

Giving M reactions and kinetic constants, N species, one initial state $X(t_0)$ and a stop time TIME, FRM performs the elementary steps summarized in Algorithm 1.

Algorithm 1 First Reaction Method

```
while $t < TIME$ do
    for j=1 to M do
        Compute $a_j(\mathbf{x})$
    end for
    Generates $r_1, \cdots, r_M$ in $U(0,1)$ and generate values for $\tau$ and $j$ according to
    Eq. 5.3 and Eq. 5.4
    $t \leftarrow t + \tau$; $\mathbf{x} \leftarrow \mathbf{x} + \nu_j$;
    print $(t, \mathbf{x})$
end while
```

5.1.2 Next Reaction Method

The *Next Reaction Method* [72] revises the first-reaction method, and it is a much more popular and efficient implementation than FRM. It uses an *indexed binary tree priority queue* $\mathcal{P}$ to find the next occurring reaction and its tentative time, and a directed graph, called *Dependency Graph* $\mathcal{G}$, to recalculate only those propensities and tentative times effectively changed after the firing of the selected reaction.

The indexed priority queue $\mathcal{P}$ consists of a tree structure storing in the nodes pairs of the form (j, τ_j). Its main characteristic is that the tree is builded and maintained in such a way that each parent node has a lower τ_j than its children. This means that the root node always contains the smallest τ_j.

Then giving the set $R = \{R_1, \cdots, R_M\}$ of M elementary reactions, let $Reactants(j)$ and $Products(j)$ be the set of species S_i reactants and products of a reaction $R_j \in R$, respectively. Let $DependsOn(a_j)$ and $Affects(j)$ be the set of the species that changing their number of molecules induce a change in the value of the propensity a_j, and the set of species that change their number of molecules when R_j fires, respectively. The dependency graph $\mathcal{G}$ is a directed graph with vertex set R and with a directed edge from R_j to $R_{j'}$ if and only if $Affects(j) \cap DependsOn(a_{j'}) = \emptyset$. Note that in this definition the self loop edges are included.

So giving M reactions and kinetic constants, N species, one initial state $X(t_0)$ and a stop time TIME, NRM performs the elementary steps summarized in Algorithm 2. Note that the procedure computes at most one uniformly distributed random number for each selected reaction, instead of the M computed by FRM. This depends mainly on two tricks. The first is the re-usage of $\tau_{j'}$ for those reactions j' not affected by the firing of a reaction R_j. The second is the application of the update formula for those reactions affected by the firing of R_j that are different from it.

Algorithm 2 Next Reaction Method

```
Generate the Dependency Graph $\mathcal{G}$
for j=1 to M do
  Compute $a_j(\mathbf{x})$
  Generate $r_j$ in $U(0,1)$, and $\tau_j$ according to Eq. 5.3
  Store $(j,\tau_j)$ into the Indexed Priority Queue $\mathcal{P}$
end for
while $t < TIME$ do
  $(j,\tau) \leftarrow Root(\mathcal{P})$
  $t \leftarrow \tau$; $\mathbf{x} \leftarrow \mathbf{x} + \nu_j$;
  for all $((j,j') \in \mathcal{G}$ do
    Update $a_{j'}(\mathbf{x})$
    if $j' \neq j$ then
      $\tau_{j'} \leftarrow (a_{j',old}/a_{j',new})(\tau_{j'} - t) + t$
    else
      Generate $r_j$ in $U(0,1)$ and $\tau_j$ according to Eq. 5.3
    end if
    Generate a sample time $\rho_j$ according to Eq. 5.4
    $\tau_{j,new} \leftarrow \rho_j + t$
    replace $(j,\tau_j)$ with $(j,\tau_{j,new})$ in $\mathcal{P}$
  end for
  print $(t,\mathbf{x})$
end while
```

Note also that even though the generation of the pair (j, τ_j) takes constant time, to maintain $\mathcal{P}$ sorted the procedure takes time proportional to the logarithm of the number of reactions M. This is again better than FRM that takes time proportional to M. More information and proofs about NRM can be found in [72].

5.1.3 Direct Method

In the DM formulation the time τ is an exponential random variable with mean (and standard deviation) $1/a0$, where $a_0(\mathbf{x}) = \sum_{j=1}^{M} a_j(\mathbf{x})$. Whereas, the index j is a statistically independent integer random variable with point probabilities $a_j/a0$. Formally, DM defines as follows the probability at time t that the next reaction in the considered volume V will occur in the infinitesimal time interval $(t+\tau, t+\tau+d\tau)$, and this reaction will be R_j.

$$P(\tau, j \mid \mathbf{x}(t))d\tau = a_j(\mathbf{x})e^{-a_0(\mathbf{x})\tau}d\tau \tag{5.5}$$

As pointed out in [75], to generate a pair (τ, j) according to Eq. 5.5 the method draws two independent uniformly distributed random samples $r_1$and r_2 in the unit interval $U(0, 1)$, taking

$$\tau = \frac{-\ln\left(r_1\right)}{a_0(\mathbf{x})} \tag{5.6}$$

and

$$j = \text{the smallest integer such that} \sum_{j'=1}^{j} a_{j'}(\mathbf{x}) > r_2 a_0(\mathbf{x}). \tag{5.7}$$

Giving M reactions and kinetic constants, N species, one initial state $X(t_0)$ and a stop time TIME, DM performs the elementary steps summarized in Algorithm 3. FRM and DM, but also NRM, are *equivalent implementations of SSA* because it

Algorithm 3 Direct Method

```
while $t < TIME$ do
    for j=1 to M do
        Compute $a_j(\mathbf{x})$
    end for
    Compute $a_0$
    Generates $r_1$ and $r_2$ in $U(0,1)$ and generate values for $\tau$ and $j$ according to
    Eq. 5.6 and Eq. 5.7
    $t \leftarrow t + \tau$; $\mathbf{x} \leftarrow \mathbf{x} + \nu_j$;
    print $(t, \mathbf{x})$
end while
```

can be proved that they substantially sample from the same density function [75].

DM pre-sums all propensity functions, and then it generates separately: the next occurring time and the next reaction to fire. While FRM computes one tentative reaction time for each reaction, and then it selects the smallest tentative time and the corresponding reaction as next occurring time and reaction, respectively. NRM simply revises FRM minimizing re-computations and maximizing re-usage for optimization.

Although, in theory the asymptotic complexity of DM and FRM coincides, in practice, DM performs better than FRM. This is mainly consequence of the smaller number of uniform random numbers required at each step. While NRM performs better than DM for systems with large number of reactions.

Before the introduction of the accelerated methods we briefly overview a famous alternative stochastic simulation method called *StochSim*.

5.1.4 StochSim: an alternative to Gillespie's Exact Methods

As with conventional deterministic methods, the implementations introduced before treats molecular species in bulk assigning to each new molecular product a different name. The population based formulation typical of SSA can be very efficient with respect to others that simulates the non reactive collisions as well. However, when molecules have many specific internal states the combinatorial complexity deriving by the possible reactions between these multi state molecules can results in a huge number of species names and reactions. In other words, changing the state of a molecule the new state and their possible reactions are identified with new names. In these cases SSA implementations becomes slow because their bottleneck operations depend by the number of reactions in the system. *StochSim* works differently. It handles multi-state variables and spatially inhomogeneous stochastic simulations. Its efficiency is not affected by the number of reactions present in the system as for SSA. In fact, StochSim considers each molecule an independent entity with its own properties, such as velocity, position and so on. This individual representation of the molecules can better describe many biochemical processes [55, 123, 155] and allow to compute a multi state molecule as a single entity. The StochSim procedure performs the following elementary steps. During initialization the method builds a look-up table that stores the probability for two molecules to react. The probabilities are computed considering formulas specific for first or second order reactions, and once processed these probabilities remain fixed for the entire execution. The look-up table consider the first reactant species in the rows, and the second reactant species in the columns. For first order reactions a special virtual pseudo-molecule is considered in the columns. Then the simulation time is divided into fixed length discrete time steps. In each time step, StochSim randomly selects two reactants. The first is taken from the real molecule set. The second is taken from the real molecule set and pseudo-molecule objects. The selection occurs sampling from using uniform distribution. If no corresponding reaction is found between the selected molecules in the look-up table, StochSim concludes that no reaction occurs in this time step.

Otherwise a uniform random number in (0, 1) is generated and compared with the probability associated to that reaction in the look-up table. If the random number is greater, StochSim concludes that no reaction occurs in this time step. Otherwise there is a reaction between these two molecules. If there is only one possible reaction, the reaction is selected to fire. Then the system is updated accordingly, and the method proceeds with the next step. Otherwise the method selects the next reaction to fire according to the classical SSA selection method, but it does that only considering the selected species.

In practice StochSim represents the only alternative to the Gillespie SSA implementations. An important question arises, what relation exists between these so different procedures and view. Shimizu and Bray showed the equivalence of their physical assumptions [155]. However, the main difference is more visible in practice. The Direct Method implementation of Gillespie's SSA is more efficient in general, especially for systems with larger number di molecules [110, 130]. But when multi-state molecules are involved in a system, StochSim can be more efficient. In any case today SSA represents the *de facto* standard in the stochastic simulation of chemical reacting systems.

5.2 Exact Methods to Accelerate the SSA

During the years, many different methods have been proposed to accelerate SSA. Many of these methods maintain exactness.

5.2.1 Exact Sequential Methods

The exact sequential methods simply speed up the way the original algorithms select the next reaction to fire. Lok et Brent [111] showed that, at least when using Gillespie algorithm, it is not necessary to introduce all the reactions and species at the beginning of a simulation. They developed a stochastic simulation software package called *Moleculizer* that uses a slightly simplified version of the next-reaction method in which reactions and species are introduced only when they are needed, and removed when they are not needed. Cao et al. proposing the *Optimized Direct Method* [42] observed that indexing reactions so that lower index values are assigned to reactions with larger propensity function, the average number of terms summed in the computation of the next reaction is minimized. Additionally, good speed up can be achieved. In particular for systems with many reactions or with large disparities in the values of propensity functions. The re-indexing must be preceded by a relatively short pre-run using the direct method in which the average sizes of the propensity functions are assessed. The *Sorting Direct Method*[175] tends to establish the desired index ordering by repeatedly interchange the index of the firing reaction with the index of the next lower indexed reaction (if there is one) whenever a reaction fires. This tactic not only eliminates the pre-run of the modified direct method, but it also accommodates any changes in the relative size of the propensity functions that might develop as the simulation proceeds. Anyway, arranging the reaction indices in order of decreasing size of the propensity functions make the linear search of the next reaction faster to compute, but it also makes that search potentially less accurate. Li et Petzold [83] have recently proposed the *Logarithmic Direct Method* (LDM). The LDM represents one of the fastest known SSA implementations [132]. Its strategy is to collect and store the partial sums of the propensity functions during the computation of the sum a_0. The index of the next reaction j can then be found rapidly by means of a binary search over those partial sums. Other logarithmic solutions have been proposed and interested reader can refer to [153] and [49]. Recent progress on constant time SSA have been proposed by Slepoy et al. [184]. Authors developed the following composition and rejection method called SSA-CR. First the method divides all reactions according to their propensity value into a constant and predefined number of groups. Then SSA-CR selects stochastically one of these groups (Composition step). By generating couple of random numbers the method iteratively checks an inclusion condition for the reactions belonging to the selected group (Rejection step) until the method finds the reaction satisfying the condition. Then it fires the selected reactions redistributing all reactions changed into the already formed groups and the algorithm restarts.

5.2.2 Exact Parallel and Distributed Methods

Although the Monte Carlo generation of SSA is not well-suited to be carried out by parallel processing [30] because the process is fundamentally sequential in nature [141], noticeably speed-up has been achieved through parallelization. There are mainly two types of implementation approaches for stochastic simulations on parallel computers: the parallelism across the method and parallelism across the simulation [164]. The parallelism across the method involves the decomposition and the separate processing of the domain [141]. When the domain regards the reaction set the approach requires that the number of reactions in the system is very large. The parallelism across the simulation is based on the fact that a large number of independent simulations (often 1000 or more) needs to be performed in order to collect statistics. However, a challenge problem in the parallel implementation of stochastic simulations is the quality of the random number generator. The independence of the generated random numbers, and the subsequent independence of simulations on different processes are the primary requirement for the success of stochastic simulations [164].

Among the methods that implement the parallelism across the method, we proposed an Optimized Parallel Implementation of FRM on Graphics Processing Units [178] that we will introduce in detail in Chap. 9. Our solution splits the reaction set, and it determines the next reaction to fire and the time by computing these samples in parallel for each subset. Ridwan et al. [141] decomposed the volume into a number of sub-volumes. Dividing the entire species into smaller independent populations, and assigning them to each sub-volume. Although this solution obtained good speed, to overcome some accuracy problem due to the violation of the well-stirred assumption, it required to introduce some synchronization point at regular time intervals. The synchronization is fundamental to maintain the entire molecules well-stirred. However, despite the fact that the methodology works for the systems under study here, it is not possible to state categorically as to whether it would work for any arbitrary system. Schwehm proposed something similar in [154] solving the already mentioned accuracy problems the method randomly exchanges molecules between neighboring sub-volumes. However, this implementation results very costly as large numbers of point to point messages must be used [141].

Among the methods that implement the parallelism across the simulations, Li et Petzold [132] proposed a solution on GPGPU. GPGPU is a very cheap, fast, widespread and easy to program as well as general purpose architecture. To deal with the requirement of the independence of the random numbers, Li et Petzold used a modified multithreaded C implementation of the famous Mersenne Twister random number generator. Other solutions implement parallel computation on clusters [107], on grid [59], on Field Programmable Gate Arrays (FPGAs) [172]. However, clusters are still relatively expensive to buy and maintain, and specialized devices such as FPGAs are difficult to program. So with low cost and high efficiency modern GPGPUs are very promising technologies for a variety of applications [132].

5.3 Approximated Methods to Accelerate SSA

Some other simulation methods sacrifices some exactness in order to achieve higher speed up. The methods can be classified into τ-leaping, Hybrid and Multiscale methods. Here, we survey them pointing particular attention on the τ-leaping because in this thesis we propose different solutions based on them.

5.3.1 τ-leaping methods.

The first τ-leaping method was introduced by Gillespie in 2001 [73]. Author identified a condition on the propensity functions, that he called *leap condition*, for which if a time period τ exists over which the propensity a_j's remain essentially constant then the number of occurrences of a reaction R_j during the time interval $[t, t+\tau)$ can be approximated by a *Poisson distributed random variable.* So rather than accounting for the time of occurrence of every molecular reaction, the method in each step select the largest τ compatible with the leap condition, it draws for each reaction R_j a Poisson distributed random sample $k_j = \mathcal{P}_j(a_j, \tau)$ and it updates the system state according to following formula.

$$x(t+\tau) = x(t) + \sum_{j=1}^{M} k_j \nu_j. \tag{5.8}$$

In the course of the years the interest in determining efficiently the value of τ that enable the maximum number of reactions that can be fired in one step has inspired improvements [74, 41]. Unfortunately, the unbounded range of values of the Poisson distributed random samples can induce so much changes that some population can be driven to become negative. This issue was recently addressed by Tian and Burrage [163] and by Chatterjee et al. [48], who have substituted the Poisson distribution with a Binomial distribution, that differently from the Poisson distribution is bounded. However, resolving the bound issue of the sampling distribution was not enough, and the negative species issue was not resolved for the general case. In fact, in applying the Binomial leaping methods to systems with multiple reactions sharing the same reactant, it was revealed that the interdependence arising from species participating in multiple reactions can still lead to negative species populations. To overcome this last case, Binomial methods for each reaction R_j defined the maximum number of occurrence of a reaction R_j before consuming one of its reagents during τ, denoted with L_j. Tian and Burrage [163] and Chatterjee et al.[48] used basically the same recipe for doing this. For each elementary reaction L_j was defined as follows. For the unimolecular reaction $S_1 \rightarrow \mathit{Product}(s)$ L_j takes x_1; for the bimolecular reaction $S_1 + S_2 \rightarrow \mathit{Product}(s)$ L_j is equal to $\min\{x_1, x_2\}$, whereas for the bimolecular reaction $2S_1 \rightarrow \mathit{Product}(s)$ L_j takes the greatest integer deriving by the division $x_1/2$. In general, for any unimolecular or bimolecular reaction R_j, L_j is assigned the value

$$L_j = \min_{i=1,\cdots,N}^{(v_{ij}<0)} \left[\frac{x_i}{|v_{ij}|} \right] \tag{5.9}$$

where the square brackets denote the greatest integer operation. Notice that the minimization in Eq. 5.9 is taken over only those species that get decreased in an reaction R_j.

In general, requiring that $k_j \leq L_j$ *can be overly restrictive* if other reactions in the system can augment the populations of some consumed reactants of R_j. For example, suppose to consider the reversible reaction $S_1 \rightleftharpoons S_2$. In this case, restricting the total number of the forward reaction to x_1, and the total number of occurrence of the backward to x_2 ignores the fact that far more of both reactions might actually occur during τ. So in absence of other reactions involving the species S_1 and S_2, actually these two reactions observe the less restrictive conditions that the number of forward reactions minus the number of occurrences of the backward reaction must be smaller than x_1, and the number of occurrences of the backward minus the number of occurrences of the forward reaction must be smaller than x_2.

Requiring that $k_j \leq L_j$ *can not be restrictive enough* if other reactions in the systems can decrease the populations of the consumed reactants of R_j. For instance, considers two or more reactions with a common consumed reactant, we must take care that the total number of firings of all those reactions should not consume more molecules of the common reactant than are available. Tian and Burrage and Chatterjee et al. addressed this requirement in two different ways.

Chatterjee et al.[48] handle the problem by generating a binomial k_j subject to the limit in Eq. 5.9, once k_j is generated the state $X(t + \tau)$ is updated by subtracting the number of reactant molecules of all species consumed in R_j in the following way $X_i(t + \tau) = X_i(t) + k_j \nu_{ij}$ for $i = 1, \cdots, N$ if $\nu_{ij} < 0$. This updating operation modifies the maximum number of firings for R_{j+1}. This step ensures that the maximum allowed firings L_{j+1} left over in executing the subsequent reaction R_{j+1} would not exceed the actually available populations. But this strategy makes its outcome dependent by the order of the reactions. Earlier considered reactions will tend to fire more often than later considered reactions; indeed, later considered reactions will not be allowed to fire at all if the earlier considered reactions have used up all the molecules of the common reactant. Chatterjee et al. tried to correct this bias by randomly changing the order in which the reactions are considered from one leap to the next.

Tian et Burrage [163] handle the problem designing a sampling technique for the total reaction number of a reactant species that undergoes two or more reactions. This technique is based on two properties of the Poisson and binomial random variables. Briefly, the properties guarantee that the sum of the samples k_j and $k_{j'}$ generated by the sampling procedure, never can exceed the total reaction number of a reactant species that undergoes the corresponding reactions R_j and $R_{j'}$. Tian and Burrage stated that this procedure can be extended to more than two reactions, although they do not give detailed instructions for doing that [39]. For instance, if in some bimolecular reaction, one of the two reactants is also a consumed reactant in a second reaction while the other consumed reactant is also a consumed reactant in a third reaction, then the constraints on the numbers of times each of those three reactions could fire would be complicated even to write down, much less develop theorems for. It thus appears that the problem of multiple reactions with common consumed reactants poses issues for the binomial τ-leaping strategy [39].

Cao et al. described a modified Poisson τ-leaping procedure that resolves the negativity problem without having to address these particular issues [39]. Recognizing that negative values of a consumed reactant are likely to arise only when the population of that reactant is small, they splits the reactions into two groups. One that collects the reactions, termed *critical*, that may produce negative populations, and the other that collects the reactions with low probability to produce negative populations, termed *non critical*. The partitioning allows to simulate the critical group one reaction at a time using DM, while the non critical by using the classical τ-leaping. Simulating at most one critical reaction at each leap reduces the probability that critical reactions can overdraw some of their reactant. Additionally, when the number of reactions expected to be fired in a leap is smaller than a constant value p, the method switches to the Gillespie's Direct Method (DM) for a number of steps q.

Rathinam and El Samad in [139] proposed two algorithms to deal with negative states called the parallel and the sequential, respectively. The parallel generates tentative samples for k_j denoted with $\hat{k}_j$. The method updates the state according to the general state update formula

$$\hat{X}(t+\tau) = x + \sum_{j=1}^{M} \nu_j \hat{k_j},$$

then if $\hat{X}(t+\tau)$ has negative components, it applies a simple bounding procedure to obtain a new non negative value for $\hat{X}(t+\tau)$ as for implicit and trapezoidal τ-leaping methods [137, 43].
The sequential instead of updating the state of the system based on the collective firing of all reactions, sequentially updates the individual reversible reaction pairs. The main idea is the same as the one proposed in [48], except that they update reversible pairs simultaneously. The choice of ordering of the reversible reaction pairs is important and it may bias the probabilities of the future outcomes. Typically this situation arises in bimodal systems (also known as bistable) where the asymptotic distribution has two modes (peaks) [139] .

Pettigrew and Resat [131] proposed MτL. This method generates the samples $k_1, \cdots, k_M$ from a multinomial distribution with probabilities $\{a_1/a_0, \cdots, a_M/a_0\}$ and number of trials N' as parameters. The key parameter N' needs to be chosen accurately in order to avoid negative populations. For this purpose N' is generated by a binomial distribution with probability $a_0\tau/N$ and number of trials N as parameters. The estimation of N is decisive because N defines the upper bound to the number of reactions that can fire in this leap in order to avoid negative states. To estimate N Pettigrew and Resat developed a simple procedure, that they called rate-limiting reactant (RLR). They formulated the problem of maximizing the value of N as an integer linear programming problem over the solution domain $R = \{\mathbf{x} + \sum_{j=0}^{M-1} \nu_j K_j \geq 0\}$, where K_j are the variables defining the samples for every reaction R_j. For each reactions R_j the procedure computes a reaction number limit K'_j based on population limits of the reactants involved and N is assigned the minimum of the K_j, for short $\min\{K'_0, \cdots, K'_{M-1}\}$. Authors state that they can prove that any random walk of length $N' \leq N$ never leaves the solution region, so the RLR method guarantees that the state generated from the individual reaction numbers produced by a multinomial distribution is non-negative. So MτL with the RLR procedure may be simply summarized as follows. For a given error control parameter ϵ, it determines the tentative time leap τ_ϵ that satisfies the leap condition. It computes N using RLR, and it sets $\tau = \min\{N/a_0, \tau_\epsilon\}$. Then M$\tau$L generates N' from the binomial distribution with $p = a_0\tau/N$ and N, and finally using the multinomial distribution with N' and probabilities $\{a_1/a_0, \cdots, a_M/a_0\}$ as parameters, MτL generates the reaction numbers $k_1, \cdots, k_M$. Follow the updating and the procedure advances to the next leap.

Recently, Cai et Xu proposed the K-leap method [37], and Auger et al. gave the R-leaping method [21], which both constrain the total number of reactions occurring during a leap to be a number K(or L) to better satisfy the leap condition, thereby improving simulation accuracy.

The idea of the multinomial distribution was also used by Cai et Xu. In this method the samples $k_1, \cdots, k_M$ are generated from a multinomial distribution with probabilities $\{a_1/a_0, \cdots, a_M/a_0\}$ and number of trials K as parameters. But K is computed differently from MτL. Authors gave three different methods to select the deterministic number N' all starting from different formulations of the leap condition. If $K = 1$, K-leap executes DM, otherwise it generates a tentative time τ according to a gamma distribution with shape parameter K and scale parameter $1/a_0$. Then all k_j's can be generated according to the multinomial distribution. The negative population issue is resolved by the strict adherence to the leap condition of the value generated for K.

Instead, Auger et al. proposed a binomial based method. They pre-select the number of firing that are expected to occur in the leap, denoted with L, and they imposed that the number of occurrence samples for all reactions be equal to L, for short $\sum_{j=1}^{M} K_j = L$. This is similar as seen before for K-leap.

Here, instead to use a multinomial distribution to generate $k_1, \cdots, k_M$ the method proposed computes each k_j by generating it by a binomial conditional distribution given the events $\{K_1, \cdots, K_{j-1}\}$. The result is invariant under any permutation of the indices. Two bounding mechanisms over L, together three alternative formulas based on three respective leap condition formulations for L, avoid the occurrence of negative populations. Given L the leap time τ is generated according to a gamma distribution with shape parameter L and scale parameter $1/a_0$.

Then Anderson developed a new adaptive τ-leaping procedure [18]. The procedure uses a leap rejection method and a postleap checks. In other words, it attempts to use a tentative leap accepting leaps that demonstrably satisfy the leap condition. It proved to be a very accurate method although it is not the most efficient one. Further, since any leap condition is ensured with a probability of one, the simulation method naturally avoids negative population values.

Peng et al. [129] presented a modified binomial leap method that improves the accuracy and the application range of the binomial leap methods. This method was proposed to be the generalization of the original binomial methods to an arbitrary number of reactions sharing the same reactant. Let S_i to be reactant of k reactions indexed by $l = 1, \cdots, k$, this method determines the maximum bound of consumed molecules S_i for each R_l. The MxN sparse matrix resulting by the sampling procedure is then used to determine the maximum number of permitted firing for each reaction R_j. The extra effort employed to the sampling procedure assures to avoid negative population whatever number and type of reactions in which a species can be reactant. However, the new sampling procedures requires non trivial computation time.

Recently, we proposed SSAL, a new method which lays in the middle between the direct method (DM) and a τ-leaping. SSAL *adaptively* builds leap and stepwise updates the system state. SSAL generates sequentially the reactions to fire verifying the leap condition after each generation. As a reaction overdraws one of its reactants if and only if the leap condition is violated, this makes it impossible for the population to become negatives, because SSAL stops the leap generation in advance. We discuss SSAL in detail in Chap. 10.

Considerable work is being done to improve accuracy of τ-leaping methods. For instance, the midpoint τ-leaping [73] is analogous to the midpoint rule for ODEs. The Poisson RungeKutta method [35], which is essentially the well known RungeKutta methods for ODEs in SDEs driven by Poisson noise [90].

Nevertheless, rigorous error analysis for τ-leaping methods have been provided by Rathinam et al. in [138]. They performed a consistency check for the original τ-leaping method showing that its local truncation error is $O(\tau^2)$ for all moments of X_t. This means that the τ-leaping method is quite consistent in the moments. They also proved that the τ-leaping is of first order weak accurate for the special case of linear propensity functions. Li [109] extended this result to general propensity functions. Whereas by adding a random correction to the original τ-leaping in each leap Hu et Li improved the accuracy demonstrating rigorously the reduction of the local truncation error to the order of τ [90].

5.3.2 Hybrid methods.

Hybrid methods was designed to simulate efficiently systems with great disparity in molecule populations. These methods combine techniques belonging to the Continuous and Deterministic and/or Continuous and Stochastic and/or the Discrete and Stochastic approaches into the same simulation framework. The idea is inspired to the fact that numerical integration of ODE's are very effective in simulating biochemical systems with high numbers of molecules, but they completely neglect stochastic fluctuations which primarily occur when only few molecules are present in the system. Instead, stochastic simulation methods reproduce those random fluctuations correctly but can only do that efficiently for systems containing relatively few molecules.

Intuitively, these methods choose a division criteria that allows to classify the reactions into fast and slow. The dynamics of the fast subsystem are assumed to evolve independently of the slow. Instead, the dynamics of the slow subsystem are in general considered dependent from the fast. This asymmetry is due to the fact that the slow subsystem cannot evolve independently of the fast because the molecular species participating to slow reactions are, in general, species whose concentrations are changed by fast reactions. Additionally fast reactions occur many more times than slow reactions and the modification induced by the occurrence of the slow reactions are negligible with respect to the change induced by the occurrence of the fast reactions. For this reason the slow subsystem has to be described by a master equation with time varying propensity functions. Moreover, synchronization between the different simulation techniques as well as information conversion are needed (e.g. concentration and molecular amount).

Then it can happen that reactions classified initially in the fast subsystem can evolve in a way that recomputing a partitioning should insert them into the slow subsystem, and viceversa. Hybrid methods that take into account this possibility have a dynamical partitioning of the system. Characterization of a hybrid method passes through which simulation algorithm it combines (SSA,ODE's, τ-leaping, SDE), whether it uses dynamic/automatic or user-defined partitioning, which partitioning criteria it adopts (population number of reagents, propensities, user-defined, heuristics ecc.), if it considers the propensities as time varying or constant [127]. Pahle [127] gave a detailed description of the main Hybrid methods, interested reader can refer to his work and references therein.

5.3.3 Multiscale methods.

Besides the coexistence of chemical species in relatively small quantities and species in larger quantities, biological systems can evolve with processes that can span several order of magnitude in time scale. The presence of great disparity in time scales, the fastest of which reach some form of stability can slows down significantly the simulation with SSA[119]. This phenomenon is known as *stiffness*. In fact, in simulating a stiff system Gillespie SSA spends very long computational time to capture the fast dynamics of the system, while the slow dynamics are simulated very slowly. The analysis of the phenomenon revealed that in stochastic simulations multiple time scales can arise in at least three different ways [86]: first, some set of *reversible* reactions occur frequently, and the remaining reactions occur rarely. This situation is analogous to reaction equilibrium. Second, some set of *irreversible* reactions occur frequently, and the remaining reactions occur rarely. Generally for this situation to be sustained some species numbers have to be large. Third, some species react so rapidly that their average number throughout the simulations is nearly zero or their average number is much smaller than the other species numbers.

To simulate multiscale systems many different methods have been proposed. Some of these methods deal with the phenomenon of rapid equilibrating reactions [86, 40, 79, 146, 145], others deal with many fast irreversible reactions [87, 74, 137], others with highly reactive intermediates [114].

Most of these methods are based on notions introduced in the deterministic counterpart, conveniently adapted to work in the stochastic context. Some examples of this assertion are the *quasi steady state assumption* (QSSA) and the *partial equilibrium assumption* (PEA). The quasi steady state is a model reduction technique used to remove highly reactive and transient species from the model [114], whereas the partial equilibrium assumes that fast reactions that reach equilibrium remain always in equilibrium. The main difference between QSSA and PEA in the deterministic context is that the former focus on the state while the latter concentrates on the reactions. The adaptation of these assumptions in the stochastic context complicated them a bit because their definitions have to consider probability distributions and sometimes chemical master equations.

Now, briefly we survey some of these methods. Rao et Arkin proposed the first method based on QSSA [136]. This method divides the species set into *transient* and *primary* species. In a deterministic context a species is transient if the net rate of change of this species is approximatively equal to zero. Otherwise it is primary. In the stochastic context Rao et Arkin state that in order to apply the QSSA they have to make two assumptions. The first states that fixed the primary species sub-set y the conditional probability distribution of the transient species z approximatively satisfies a specific definition of the chemical master equation. The second states that the net rate of change for the conditional probability distribution of the transient species is approximatively equal to zero. Resolving, the method finds a stationary conditional probability density for z. Then at each step simply generates a conditional random variable $z(t)$ from the statinary distribution $P((z(t)|y(t)))$. The resulted samples change the value of the state for these transient species. Then the method computes the propensity functions of the primary species using this new state, and then proceeds normally executing the Gillespie's Direct Method until a the next generation.

A different approach is accomplished by Cao et al. with the slow-scale SSA (ssSSA)[40]. The algorithm proceeds as follows. The first step consists to make a provisional partitioning of the reactions into fast and slow subsets denoted R^f and R^s, respectively. The division occurs according to the value of the propensity functions. In the second step the method divides the species into fast and slow denoted with S^f and S^s. Any species whose population gets changed by a fast reaction is classified as fast, the rest are slow species. The third step defines a virtual fast process $\hat{X}^f(t)$ as the fast species populations evolving under only the fast reactions R^f. Next the algorithm applies the *stochastic stiffness condition*. This condition first requires that the virtual fast system $\hat{X}^f(t)$ must be stable. Second check that the limit $\hat{X}^f(t) \rightarrow \hat{X}^f(\infty)$ must be effectively accomplished in time, and it checks that it is small compared to the expected time of the next slow reaction. This gives a formal specification of the separation between the time scale of the fast and slow reactions. If the stochastic stiffness condition is satisfied, the ssSSA invokes the slow-scale approximation. The slow-scale approximation states that the method can ignore the fast reactions and it can simulate one slow reaction at a time. The propensity of each slow reaction needs to take into account the modifications to the state induced computing the asymptotic virtual fast process $\hat{X}^f(t)$ [44].

Samant et Vlachos [146] divide the reactions and the species into fast and slow according to the value of the propensity functions. Then the method performs SSA for the fast reactions over a number of preselected events (window). At the end of the window the method checks its own equilibrium condition. If the condition is not satisfied than it performs again SSA for the fast reactions, otherwise it runs SSA over a longer window to collect equilibrium statistics and evaluate the equilibrium state. Then taking into account the equilibrium state it selects the next slow reactions from the slow reactions. The method updates the state, and it starts a new step by recomputing the propensity functions, re-dividing the reactions and the species.

Haseltine et Rawlings tackled the issue of stiffness approximating the fast species either deterministically or via the Langevine equation and the slow one via the SSA

[87]. In [45] Cao et al. introduced a new stochastic formulation of the partial equilibrium assumption and they provide a test on a non trivial biological model. Whereas Weinan et al. proposed a method based on averaging theorems for Markov processes which allows to identify the fast and slow variables in the system and the effective dynamics over the slow time scale [61].

However, the complex system may not always remain in a partial equilibrium state [46]. When the system is not in partial equilibrium, it is necessary to simulate the fast reactions accurately to reflect the corresponding dynamical changes. However, when the fast reactions reach the partial equilibrium state, it is more efficient to focus on the slow-scale reactions. This can be achieved by keeping dynamic lists of fast and slow reactions, verifying equilibrium conditions during the simulation. However, the frequent house-keeping operations can be computationally expensive and they can impact the simulation efficiency. Moreover, when the system exhibits modes between fast and slow, the partial equilibrium method is not applicable. Cao et al proposed a new adaptive method [46]. This method dynamically switches between implicit and explicit tau-leaping methods without explicitly distinguishing the fast and slow scales. To do that the method compares the step-sizes given by the implicit τ-selection formula and the explicit τ-selection formula.

Generally speaking, the model reduction methods have been applied to a number of stiff systems (e.q. Michaelis-Menten system) with great speed up (2 or three order faster) with respect to SSA. However, some issues deriving from the generalization to nonlinear kinetics, the *one time* system partitioning and the simultaneous contribution from both population size and rate constants have to be resolved [146]. Then we also noticed that only very few tests have been conducted to non trivial systems and to non stiff systems, or for systems with no great separation in time scales.

Recently, Cao et Petzold in [44] highlighted also that the adaptive method has difficulties in effectively handling the situation when a species with a small population is involved in a fast reactions. Situation that also results in low efficiency for hybrid methods [44]. Then authors combined the ssSSA and the adaptive τ-leaping [44]. They designed a method to automatically detect the fast but stable reactions. They prepose a novel idea based on an automatic partitioning of the reactions into reversible reaction groups.

Instead, the HyMSMC method [147] blends stochastic singular perturbation concepts, to deal with potential stiffness, with a hybrid of exact and coarse-grained stochastic algorithms, to cope with separation in population sizes. In addition, authors introduce the computational singular perturbation (CSP) method as a means of systematically partitioning fast and slow networks and computing relaxation times for convergence.

In conclusion the stiffness problem and the solutions proposed in the stochastic simulations represent a very interesting field, however in this thesis we do not deal with them directly.

5.3.4 Accuracy Measurement

Approximated simulation methods can achieve great efficiency and give a close approximation to the SSA method. In this section we address the problem of measuring the accuracy of an approximated methods.

One possibility for measuring the error can be compute the errors in solution moments such as the mean and variance. Sometimes, these low-order moments are not enough. For instance, considering a bi-stable system (e.g. Schlögl model [34]) the mean and the variance are not relevant. So in [34] Cao et al. seek a more precise quantitative measurement introducing two distribution distances. The first is the *Kolmogorov distance*, defined to measure the distance between cumulative distribution functions (cdf). The second is the density distance area, defined as the L_1 distance between the probability density functions (pdf). Due to the limited number of Monte Carlo simulations these two continuous distances have been adapted to consider discrete values. So, an *empirical distribution function* (edf) is used to measure the cumulative, and a histogram is used to measure the pdf.

Let N independent random variable $x_1, \cdots, x_N$ having the same distribution, the edf is defined as follows.

$$F_N(x) = \frac{1}{N} \sum_{j=1}^{N} \kappa(x - x_j) \tag{5.10}$$

where

$$\kappa \;= \begin{cases} 1 & \text{if } x \geq 0, \\ 0 & \text{if } x < 0. \end{cases} \tag{5.11}$$

The $F_N(x)$ gives the fraction of points smaller than x. Supposing that all the observations fall in the interval $I = [x_{min}, x_{max})$ and letting $L = x_{max} - x_{min}$, we are able to divide the interval I into K subintervals, denoting the subintervals by $I_i = [x_{min} + \frac{(i-1)L}{k}, x_{min} + \frac{iL}{k})$, the histogram function h_X is reported in Eq. 5.12.

$$h_i(I - i) = \frac{K}{NL} \sum_{j=1}^{N} \chi(x_j, I_i) \tag{5.12}$$

where

$$\chi(x_j, I_i) \;= \begin{cases} 1 & \text{if } x \in I_i, \\ 0 & \text{otherwise} . \end{cases} \tag{5.13}$$

The histogram $h_i(I - i)$ gives the fraction of points falling into the interval I_i. Starting from the definitions of $F_N(x)$ and $h_i(I - i)$ the distribution distances are defined as follows.

For two random variables $X = X_1, \cdots, X_N$ and $Y = Y_1, \cdots, Y_M$ with edf F_X and F_Y respectively, the Kolmogorov distance is

$$K(X, Y) = \max_{-\infty < x < +infty} |F_X(x) - F_Y(x)| \tag{5.14}$$

For two groups of samples X_i and Y_j, the *histogram distance* is

$$D_K(X,Y) = \sum_{i=1}^{K} \frac{|h_X(I_i) - h_Y(I_i)|L}{K}. \tag{5.15}$$

A notable difference between the Kolmogorov and the Histogram distances is that the histogram distance considers the absolute value of the density difference, whereas the Kolmogorov distance considers the sign. Because the signed differences may cancel each other, the Kolmogorov distance may underestimate the difference [34]. In any case, both the Kolmogorov distance and the histogram distance can be used to measure accuracy.

Now, in stochastic context with a limited number of simulations, two groups of samples X and Y, even though they are taken from the same distribution can be different [34]. In the estimation of accuracy we need to take into account this distance, called *self distance*. For two sets of independent samples $X = X_1, \cdots, X_N$ and $Y = Y_1, \cdots, Y_M$ that follow the same distribution, self distance is defined as the distribution distance between X and Y. The Kolmogorov self distance is denoted with $K(X,Y)$, whereas the Histogram self distance with $D_K(X,Y)$. The self distance is a random variable with mean and variance. The bounds for the mean and the variance of the Histogram and the Kolmogorov self distance are given in [34]. For sufficiently large N and M, the mean and the variance of histogram self distance $D_K(X,Y)$ are bounded by $\sqrt{\frac{2K}{\pi}(\frac{1}{N}+\frac{1}{M})}$ and $\frac{(\pi-2)K}{\pi}(\frac{1}{N}+\frac{1}{M})$, respectively. Whereas for sufficiently large N and M, the mean and the variance of Kolmogorov self distance $K(X,Y)$ are bounded by $\sqrt{\frac{\pi}{2}\log 2\sqrt{(\frac{1}{N}+\frac{1}{M})}}$ and $(\frac{\pi^2}{12} - \frac{\pi}{2}\log^2 2)(\frac{1}{N}+\frac{1}{M})$, respectively. The values of the self distances have to be interpreted as follows. The closer to the self distance a distance value is, the more accurate the method who has generated that samples will be.

StochKit provides a simple Matlab DataAnalyzer package to generate and plot statistical information from ensembles [105]. In particular, the package provides functions to compute the distribution differences. In this thesis any measurement of the accuracy is made through distribution distances implemented in the Matlab DataAnalyzer package. The two group of samples necessary for the computations are collected as follows. Giving a model with N species and M reactions we run a number $RUNS$ of independent simulations with our approximated method. Then we compute the same number $RUNS$ with SSA. At the end of each simulation we store the value $X_i(TIME)$ for a given species S_i at a given final time TIME. Completed the $RUNS$ simulations we have two groups of samples necessary to compute the distribution distances and we have enough information to compute the bounds for the relative self distances.

5.4 Chemical Kinetics with Different Hypothesis

Recently, more sensitivity about the applicability of the Gillespie's stochastic formulation of chemical kinetics deal to new alternative formulations that relax some of the Gillespie's hypothesis. In this section we briefly overview the basic ideas of these approaches introducing some representative method.

5.4.1 Chemical kinetics with Spatiality

The Gillespie's Stochastic formulation assumes that the system is well stirred. This assumption holds in case of an equilibration of reactants between all positions in the system volume occur on a much faster timescale than the chemical reactions. Since diffusion of molecules in a living cell is considerably slower than in the test tube [64], the condition of spatial homogeneity is expected to be violated in most cases. In fact, many important intracellular processes such as cell division [89], morphogenesis [165], some type of chemotaxis [162] and metabolic and signalling pathways as it is known depend on spatial heterogeneity. Molecular Dynamics, Partial Differential Equations, Brownian Dynamics, Lattice based methods and Spatial Gillespie methods have been proposed to incorporate spatial effects in biochemical networks, as reviewed in Takahashi et al. [160]. However, currently there is no single model capable of efficiently coping with the broad range of spatial, temporal and concentration scales commonly found in biochemical networks. For instance, the highly accurate microscopic methods are too computationally demanding to simulate full pathways, while macroscopic ODE models cannot cope with spatial and stochastic phenomena. For this reason mesoscopic models may represent a plausible alternative approach and a compromise between computational efficiency, spatial and stochastic accuracy.

One approach for mesoscopic models is to tackle inhomogeneities, or spatial localization of reactions, by dividing the space into smaller homogeneous sub-volumes. These sub-volumes are considered to be well-stirred systems for which efficient well-stirred reaction methods can be used, either stochastic or deterministic [169, 54, 156, 63, 22].

Stundzia and Lumsden [156] proposed the first treatment of diffusion in the stochastic method for non linear reaction-diffusion processes. They extended the Gillespie's Direct Method to reaction-diffusion processes in spatially inhomogeneous systems, and they proved equivalence of the algorithm with the reaction-diffusion master equation. The bounding surface of the volume is taken to be totally reflecting barrier, impermeable to diffusion.

ELf and Ehrenberg [63] extended the Next Reaction method seen in Sec. 5.1.2 to be used in 3D sub-volumes. They called this method *Next SubVolume Method* (NSM). The sub-volume sizes are determined such that all reactive molecular species, represented as point particles, are almost uniformly distributed in each sub-volume space. This is done by ensuring that the diffusion of reactants in a sub-volume takes place more frequently (e.g., more than 100 times) than their respective reactions.

At each time-step, each molecule can either react in its current sub-volume or diffuse to an adjacent one. The diffusion probability at each time-step is obtained by mapping the bulk diffusion constant in Fick's law using the Green's function. Similar to the original Next Reaction method, the computation time increases only with the logarithm of the number of sub-volumes in the system. Nonetheless, it is not possible to reproduce crowded conditions because volume exclusion from both reactive and non-reactive crowder molecules cannot be represented explicitly when they are depicted as point particles.

Jeschke et al. provided the parallel and distributed implementation of NSM [92], while Marchez-Lago and Burrage proposed a new coarse grained modified version of the NSM that uses the binomial τ-leaping [113]. Bernstein [26] has extended the Gillepsie's Direct method for simulating reaction-diffusion systems on irregularly spaced sub-volumes (or Cartesian meshes in 2D). The system may have inhomogeneous diffusion coefficients, including those with discontinuities.

The Gillespie Multi Particle Method (GMP) [142] is another approach for simulating reaction systems with an inhomogeneous distribution of discrete number of dimensionless, uniformly distributed particles using cubical lattice. In contrast to NSM, diffusion events take place through an operator split scheme at predetermined times, lifting the diffusion to the macroscopic level. The geometry of a membrane is represented by a number of lattice sites. These membrane sites also hold cytosolic molecules, enabling, membrane-cytosol reactions. During initialization, the first diffusion event time is calculated for each species. In each iteration, the event with the smallest time stamp is selected and Gillespies Direct Method is used to simulate reactions between the last and the next event time. The execution of the next diffusion event is then performed locally in each sub-volume, distributing all entities of the species assigned to the event randomly among its neighbors. Therefore, in contrast to NSM, the GMP method abandons the idea of single entity diffusion events between two sub-volumes and performs simultaneously bulk diffusions of species entities in all sub-volumes.

5.4.2 Chemical kinetics with Delayed Reactions

In the Gillespie's stochastic formulation it is assumed that all reactions occur instantly. While this is true in many cases, it is also possible that some chemical reactions in living cells takes certain time to finish after they are initiated. Thus, the product of such reactions will emerge after certain delays [36]. Delay processes are ubiquitous in the biological sciences but sometimes they can also be ignored. For instance, when delays in biochemical reactions are small compared with other significant time scales characterizing the system. If the time delays are of the order of other processes or longer, they cannot be ignored and taking the delays into account can be crucial for the accurate description of transient processes. To this end, Bratsun et al. showed how time delay in gene expression can cause a system to be oscillatory even when its deterministic counterpart exhibits no oscillations [28].

They also proved how such delay-induced instabilities can compromise the ability of a negative feedback loop to reduce the deleterious effects of noise.

To deal with delays in the stochastic simulation framework has required to modify the Markovian procedure typical of SSA. In fact, advancing the state one reaction at a time, the simulation can result in wrong evolutions due to the non-Markovian nature of the reactions with delays. When some of the reactions are non-Markovian, Bratsun et al. [28] modified the original version of the direct method algorithm as follows. At each step the method generates the next reaction and time according to Eq. 5.6 and Eq. 5.7. If the next time event is t^* but the selected reaction is delayed, the reaction is placed in a stack and it will actually be completed at time $t^* + \tau$. If, however, the chosen reaction is Markovian, the time of the next reaction t^* is compared with the times of the previously scheduled delayed reactions. If none of those scheduled reactions occur before t^*, the time is advanced to t^*, the state is updated according to the chosen non delayed reaction, and the process repeats. If, however, there is a delayed reaction scheduled for completion at $t_d < t^*$, the last selection is ignored, time advances to t_d, the scheduled delayed reaction is performed, and the selection process repeats.

Barrio et al. proposed an exact [36] generalization of the Gillespie's SSA with delay, named *delay stochastic simulation algorithm* (DSSA) [23]. Different versions of DSSA have been discussed by Barrio et al in the supplementary material. Authors identified the following simulation aspects to characterize these versions:

1. Waiting time. In the SSA the time between two reactions is regarded as the waiting time until the next reaction occurs, while reactions happen instantaneously. In the DSSA one can *ignore* the waiting time until the reaction considering this time included in the delay time. Otherwise one can compute the waiting time and the delay time separately.

2. Time steps in the presence of delayed reaction updates. The problematic situation is the following. Suppose that the algorithm selects a reaction for firing but a delayed reactions can fire in the same time step. One can ignore the selected reaction that should be updated beyond the current update point. But this can be considered as changing the stochastic path. Otherwise we can ignore any changes of the systems state due to the delayed reactions within this time step. But in this case we loose the property of only one reaction per time step.

3. Updating delayed consuming reactions. The systems state at the moment of selection of the delayed reaction can be very different to the state at the moment when the delayed reaction is updated [23]. When in the period between selection and update of a delayed consuming reaction other reactions occur consuming the same reactants, by updating the delayed reaction the molecular number of those reactants can become negative. Therefore, reactants and

products of delayed consuming reactions must be updated separately namely when the delayed reaction is selected and when it is completed, respectively.

The first DSSA considers the waiting time included in the delay time (1). It updates a delayed reactions ignoring a reaction (2) if the time step would pass the update points of a delayed reaction. It works only for non-consuming reactions (3) since there is only one update when a delayed reaction is due.

The second DSSA computes the waiting time and the delay time separately (1). Moreover, the delayed reactions that are scheduled at time points passed by a simulation step are updated all together with the latest reaction (2). It works only for non-consuming reactions (3).

The third DSSA include waiting times (1) as the second algorithm and the updates of only one reaction per step ignoring the reaction that is selected for the time step (3) as the first algorithm. However, it still does not run with delayed consuming reactions. This deficiency is remedied with the fourth DSSA.

All experiments in the paper of Barrio et al. run with the third DSSA. Recently some other works have been proposed. Starting for the fact that delay can be deterministic but also stochastic, Roussel and Zhu [144] generalized previous DSSA by proposing an algorithm which allows both multiple delay and stochastic delay times. Finally, Anderson in [19] proposed an efficient modified NRM for simulating chemical systems with time dependent propensities and delays.

Part III

Case Study for biochemical processes simulation

Chapter 6

Modeling perceptive functions: the photoperception in *Halobacterium salinarum*

The capability to perceive the surrounding world and to react to environmental stimuli is one of the most important characteristics of living organisms, already present at the early stages of the biological evolution. The *Halobacterium salinarum* has been studied since thirty years, because of the fascinating and very complex behavior it shows when exposed to light. To explain these complex behaviors different qualitative models have been proposed.

The aim of our work is to apply a systems biology approach to the most accredited model, by rewriting it as a stochastic system of biochemical reactions and to explore its dynamic time course. This will enable us to test if this model can actually explain correctly the light responses obtained by exposing *H. salinarum* to different light stimuli.

We formed a multidisciplinary group, that we called BioLab, to cope with this problem. In this chapter we introduce the problem listing the behaviors observed in wet-lab experiments and the main models.

6.1 Overview

H. salinarum is an halophile organism belonging to the Archæa domain that lives in environments of high salt concentration like the Dead Sea and salt evaporation ponds. It has multiple flagella at the same pole, which are used synchronously to move in a specific direction.

In its habitats, solar radiation is very intense, so *H. salinarum* needs to be able to detect light intensity and exhibit adequate motile responses.

H. salinarum swims by rotating its flagellar bundle; when changing from clockwise to counterclockwise and vice versa, the archæon 'tumbles' and changes its swimming direction. In normal conditions of light intensity and nutrients concentration, the archæon tumbles with a random reversal frequency, switching every 5 to 50 seconds. Variations of different kinds in the environmental conditions result in a change in this frequency, in order to avoid damaging situations or to make use of newly available resources [7].

These changes in the swimming behavior happen by mediation of a series of archæal rhodopsins embedded in the cell membrane. There are, in fact, four different light-activated proteins in *H. salinarum*: bacteriorhodopsin, halorhodopsin, sensory rhodopsin I (SRI) and sensory rhodopsin II (SRII). Only the last two play a proper photoreceptor role, as the first two are classified as light-activated ionic pumps.

An increase in the reversal frequency is considered a *repellent* response, because the cells are induced to flee from the site where the stimulus was felt. This can be caused by the presence of harmful molecules in the environment, or dangerous light wavelengths that may cause damages to the cell (like UV light). An *attractant* response is instead an increase in the time elapsed between a reversal and the next in order to allow the cell to use the nutrients or the light energy for its growth.

Different light wavelengths interact with the sensory rhodopsin I and II, associated to transducer complexes called Htr (Halobacterium transducer proteins). The light stimuli are therefore converted in biochemical signals that enter in a pathway, resulting in a modulation of the flagellar activity: when the receptor absorbs photons of a specific wavelength, their energy allows a transition within the receptor to new states. These *signaling states* interact with the transducers; the stimulus arrives in two different sites and causes the activation of a histidine kinase by autophosphorilation, and a change in the susceptibility to methylation of a second site. The histidine kinase loses its phosphate by activating a cytoplasmic protein that causes a transient alteration of the switching probability of the flagellar motor. The second site then causes a return of the transducer to its initial activity [4].

In an aerobic environment, *H. salinarum* tends to avoid sunlight to prevent oxidative damage; in such a situation, the only rhodopsin produced and exposed on the cell surface is SRII (also called phoborhodopsin). SRII absorbs blue-green light, the energy peak of the solar spectrum; its activity allows the cell to detect high light intensity and to flee from this dangerous situation.

When an oxygen tension drop occurs, SRII's synthesis stops and BR and HR are produced instead. The first pumps out protons and the second pumps in chloride ions, contributing to the motive force needed to synthesize ATP and preventing cytoplasmic alkalization.

In this situation, SRI is produced too. SRI mediates an attractant response to orange wavelengths, allowing the cell to stay in the light more, in order to take advantage of the light energy available to produce ATP. At the same time, SRI prevents damage from too high light energy, thanks to a photointermediate that mediates a repellent response from blue light (with an absorption peak in the near-UV wavelengths). SRI therefore allows an attractant response to light only in absence of potentially damaging photons.

6.2 SRI's Photocycle

In the last thirty years, a very large collection of experimental data has been acquired, and some qualitative models have been proposed to explain the different response patterns shown after light stimulations. To explain this complex photobehavior, in these proposed models the main role is covered by the photoreceptor molecules, which have shown to perform photocycles.

In particular, the two most reliable models proposed to explain the photobehavior of *Halobacterium salinarum* are focused on the properties of the SRI, which can assume several spectroscopic states arranged together in a *photocycle*. These two models differ each other for the number of signaling states they identify among the different spectroscopic states. The first one has been proposed by Marwan *et al.* [7] where the SRI has four different spectroscopic states. In this model, a state is responsible for the increase in switching, and another for the decrease. The other two states do not have signaling activity (Fig. 6.1). The second proposed by Hoff

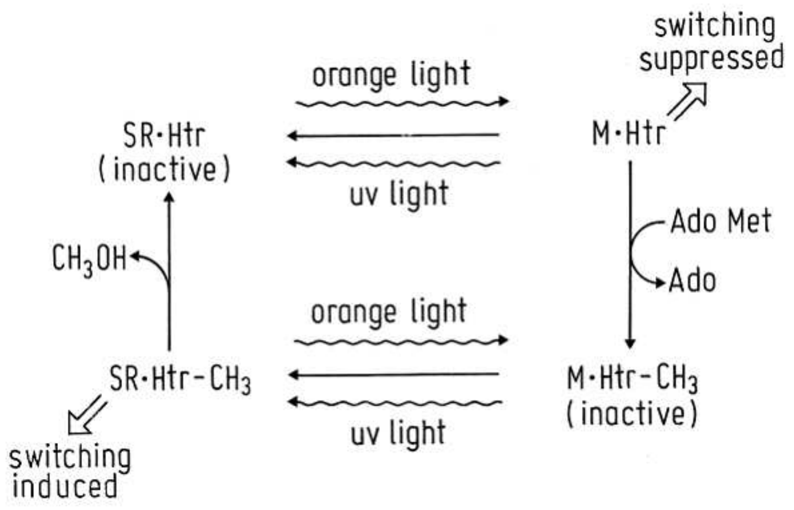


Figure 6.1: The four-state model, from [7]

et al. [4] has seven spectroscopic states (Fig. 6.2).

This model assumes two distinct cycles for the responses to orange and blue light. For each cycle there are two signaling states, one inducing and one suppressing reversals. This has been obtained after experimental studies of the photocycle behavior.

SRI's fundamental state is called SR_{587} for its maximum absorption wavelength. When a photon is absorbed, it starts a cyclic series of transitions with the conversion of the receptor in three intermediate states, called S_{610}, S_{560}, S_{373}, of which the only long-lived is the last, which then reconverts to SR_{587}. If blue light is present, S_{373} absorbs it and reconverts more rapidly via another intermediate called $S^{b}{}_{510}$. This

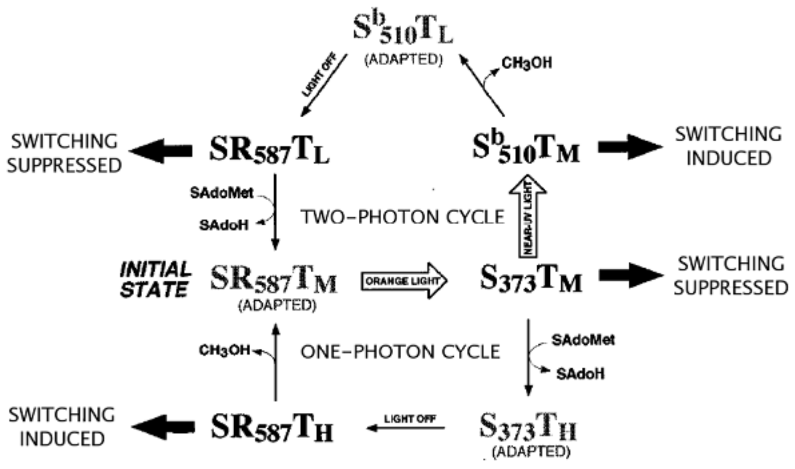


Figure 6.2: The seven-state model, modified from [4]

photochromic effect and the opposite responses it mediates allow the cell to adequate its behavior in different light conditions. Both S_{373} and $S^b{}_{510}$ decay thermally to SR_{587}, though with different constants [4].

It has been shown that the attractant response to orange light is proportional to the concentration of S_{373}, indicating this intermediate as the signaling state for this kind of response. The S_{373} produced during on a continuous orange light stimulus is also proportional to the repellent response from blue light, thus giving a double role to this intermediate.

In the case of simultaneous stimulation with both blue and orange light, however, although S_{373} concentration increases, a strong repellent response is registered. This suggests the existence of a second signaling state, specific for the repellent response. The only intermediate with a lifetime compatible with this function was shown to be $S^b{}_{510}$ [11].

6.3 Phototaxis

When unstimulated, *H. salinarum* reverses its swimming direction randomly about every 5 to 50 seconds. Light stimuli and concentration gradients alter this reversal frequency, causing different kind of responses in its behavior.

An increase in orange light causes an attractant response: reversals are suppressed for a short period of time, during which the cells use this specific wavelength for their growth. On the other hand, blue light and UV radiations determine a repellent response increasing this frequency. UV light in fact is very dangerous for the cell, since it can cause mutations in DNA and alter the nature of the protein-protein interactions.

This reflects what happens in the case of an orange light spatial gradient. The cells switch their directions less frequently when they are swimming up the gradient, while they reverse more often when they're swimming down the gradient, resulting in a net movement towards the higher light intensity. Likewise, a blue light gradient determines an opposite response, with a net direction towards the lowest intensity areas.

The reason of this behavior is a change in the steady state concentration of SRI's signaling states. Attractant signaling states inhibit the histidine kinase's activity, while repellent signaling states promote it. However, even when a stimulus persists and the signaling states' concentration results altered, the cells' response are transient, which means that a second signal inducing adaptation is involved in this mechanism.

Indeed, some seconds after the stimulus begins, the cells show an adaptation and return to their pre-stimuli reversal behavior. As seen before, this feedback is mediated by the activation of CheB[1], which brings the transducer to its initial state of methylation and stops the transduction, whereas the photocycle keeps responding as before.

At the end of a stimulus, an opposite response is registered. For attractant stimuli it means a sudden peak in the reversal frequency; for repellent stimuli the result is instead a decrease.

All the cell transient responses are therefore results of interactions between excitation and adaptation signals [4]. Depending on the photons wavelengths, the possible responses are:

- A primary response to an increase of S_{373} after a step-up in orange light;
- A deadaptation response to the decrease in S_{373} concentration after a step-down in orange light;
- A primary response to an increase in ${S^b}_{510}$ concentration after a blue light step-up;
- A deadaptation response to the decrease in ${S^b}_{510}$ after a blue light step-down.

The primary responses are caused by a transient increase in the signaling state concentration, which promote the histidine kinase activity. The deadaptation responses are caused by the decrease in the signaling states' concentration, probably due to a role of the methylation level in the transducer, that changes transiently after a light step-down, determines an independent action on the kinase and results in a change in the reversal frequency following the step-down.

[1]CheB is a carboxylmethyl esterase which acts a pivotal role in responsiveness and feedback control, restoring the methylation levels of the transducer to its pre-stimulus values.

6.3.1 Complex stimuli

More complex stimuli patterns determine peculiar and interesting responses. Some patterns are reported to cause integrative responses when delivered in particular conditions, while others cause responses opposite to the usual [3].

A first example is the so-called *paradox effect*: when a single orange pulse of 1s or less is given to a cell adapted in the dark, it causes no changes in the cell's behavior, or determines a weak attractant response at most. But if the pulse is given shortly after an orange step-down, it promotes reversals. To cause this behavior, the second stimulus must be a very short pulse, because it has been shown that step-ups do not promote reversals in this condition.

Another interesting case is the stimulation of the cells with a train of short impulses. In this condition too, when given alone the stimuli do not elicit a response. But if every 3-4 seconds a 1-second stimulus hits the cells, the first stimuli do not elicit a response, but the following ones determine an increasing response after each pulse, up to an asymptotic value. When the interval between one pulse and the following is too short to promote a response, only the last pulse elicit reversals, with a strong increase in the population's reversal frequency. The cells do not behave like if a series of pulses is given, but show a response very similar to the one for orange step-ups, with a strong repellent response at the step-down.

The cell is also able to respond to orange and blue lights given simultaneously. The result, which is the same as to give white light, is a repellent response. The response to blue light seems to prevail on the response's mechanisms to orange light. As said before, the discovery of this behavior led to the assumption that in the orange and blue light responses must be involved different states of the photocycle.

As shown in these and other responses to complex stimuli, within this ensemble of photoreceptor and transducer happen important events of integration and memory. The exact nature of these events is not clear, and their understanding and description is an open challenge.

Chapter 7

QDC: Quick Direct Method Controlled

Most of the algorithms reviewed in Chap. 5 are used to simulate biochemical systems as *isolated universes*: there are neither external forces nor events that can alter the time course of the system evolution. This actually is a great limitation. In fact, one would be able simulate the time course of a real metabolic experiment. Experiments differ from simple simulations because in the experiments the operator performs perturbing actions on the observed system with the aim of investigating how the system responds to a stimulus. In the last few years, different simulators have been proposed to take into account the possibility to perform actions on the simulated biochemical systems, in order to have an *in silico* representation of metabolic experiments performed in wet-lab [133, 58, 143]. These simulators enhance their syntax in order to describe more actions, system states and events that can take place in the extended framework of metabolic experiments. They present a great variability with respect to the events they can simulate and the actions they can perform on the simulated systems; nevertheless, in our opinion, they do not cover all the most frequent events that can take place in signaling pathways or metabolic experiments, and their syntax is often very complex for people with a biochemistry or biology background. We wanted to develop a simulator able to give a quick and easy (i.e., with a language similar to that of biochemistry) representation of metabolic experiments with a comprehensive description of the most frequent experimental controls used. Within this perspective we designed QDC (Quick Direct-method Controlled).

7.1 The approach: from biochemical systems to metabolic experiments

Usually, real metabolic systems are perturbed by signals and stimuli, either endogenous[1] or exogenous[2], that can alter the system status, by changing the concentration of some chemical species, as well as by supplying new species or changing the rate of some reactions. For example, a signaling pathway changes its status dependently on the signal molecules presence/absence. Moreover, metabolic experiments are often designed to alter the behavior of a system by adding new chemicals to it, or by enhancing or reducing the rate of a give reaction by means of agonists or inhibitors. In photoperception experiments, the light switching on/off dramatically changes the rate of the reactions that involve the activated photoreceptor molecule. In metabolic drugs testing experiments, a drug is supplied to the system at a given time and at a given concentration. In the present chapter we propose a new simulator, called QDC (quick direct-method controlled), that has been designed to simulate metabolic experiments through the specification of several control statements that simulate different actions the operator performs on the real metabolic system during wet lab experiments. QDC's syntax has been developed to be as close as possible to the standard syntax of biochemistry, to allow biologists to have a natural confidence with this tool. The list of control actions that is possible to specify under QDC represents the most common events that can take place in signaling pathways or metabolic experiments; in particular QDC allows to:

a) indicate how many molecules of each species are added to the system and at what time they are added (when the time is zero, this value represents the initial quantity of that species);

b) represent the rate of a reaction with a parametric variable, that can change its value when certain conditions are satisfied (e.g., after a certain time);

c) specify zero-order reactions that can be used to simulate uptaking systems and other possible experimental set-up that supply some chemical species with a constant rate;

d) specify immediate reactions; here the term *immediate* means that these reactions do not have a kinetic law, but they take place immediately, once the stoichiometric conditions of the left side of the equation are verified.

The difference between point a) and point c) is that in the former case a given number of molecules is added at a given time, while in the latter case molecules of a given species are added regularly with a certain rate. This can be important, for example, when it is necessary to distinguish between two ways to collect food: it

[1]Generated from within the system.

[2]Action or object coming from outside a system.

is, in fact, already known that the behavior of the glycolysis in *E. coli* depends on glucose availability. When glucose is supplied with a given frequency, oscillations arise in the time course of fundamental metabolites; these oscillations represent an emerging property of the systems [161, 170, 51]. Instead, when glucose is available in a virtually infinite number of molecules, no oscillations are observed. Molecular biology of the cell and laboratory experimental protocols show a very wide variety of similar situations, raising the challenge to have a simulation environment able to reproduce such a emergent property. QDC is not designed to take into account cellular compartments or membranes: the simulated biochemical system is imagined as well-stirred and compartment-free. Nevertheless, QDC allows the user to simulate molecular flows in/out of the simulated system by using the NULL operator. When it is placed at the left side of the equation in 0-order reaction, it is used to simulate the intake of molecules. When NULL is placed at the right side of either a 1st-order reaction or an immediate reaction, it is used to simulate the excretion of chemical species from the simulated system. In this way, the simulated system is not considered as an "isolated universe", but as an open system which can import/export material from/to a generic "outside". From a computational point of view, QDC uses a very efficient implementation of Gillespie's direct method to simulate biochemical systems, as its computational performances are comparable or even better than those of the most popular simulators, as detailed in the following.

7.2 Our tool: From input to output, SBML Import/Export and GUI

7.2.1 QDC's syntax specification and input file structure

The definition of input system and events must be specified using the QDC input format. QDC's input is represented by an ASCII file, written in a home-defined format, where the syntax to express reactions is very similar to that of biochemistry (see Fig. 7.1). The input file is structured in subsequent blocks, separated by a blank line. Each block contains a different category of information that we briefly summarize as follows (see Appendix A for a more rigorous description).

The first block contains the name declaration for each biochemical species that will take part in the system; each valid name is an alphanumeric string with possible underscore character starting with a letter. This last requirement is to avoid confusion with the stoichiometry values of a reaction. One row can host several names separated by a comma. Some names are reserved for internal program usage or for SBML import/export; in particular: `NULL, null, parameter, event, reaction, addition, volume`. These terms cannot be used in the user-defined input code.

The second block declares the volume (measured in liters) taken by the system, as it is required for transforming reaction rates (deterministic) into stochastic propensities for chemical reactions whose order is greater than 1. The line statement is of the form: `volume,` $<$ *value* $>$.

The third block contains all the biochemical reactions hosted by the system. The general reaction syntax is:

$$r, \texttt{A + B} > \texttt{C + D + ...}$$

where `A, B, C, D` represent any allowed symbolic name of a chemical species. The numerical value in input for the reaction rate is the deterministic one that QDC will transform into the stochastic propensity, accordingly with the reaction order. The usual arrow of the chemical equation is represented by the sign “>”. QDC is able to manage and simulate reactions of different orders:

- 0-order reactions. These reactions specify the “creation” of a given species: they are very useful for representing the effect of an uptake system that supply a certain species at a given rate. The syntax is of the type:

  $$r, \texttt{NULL} > \texttt{A} + \ldots$$

  As the term `NULL` (even lowercase `null`) is a reserved name, it must not be declared among the chemical species. For the same reason, it is not allowed to assign a number of molecules to it. The rate of 0-order reactions depends neither on the molecule concentration nor on the volume do they occupy.

- 1st-order reactions. These reactions describe processes like isomerization, auto-splicing, decay, fission and similar, where the left term of the chemical equation involves a single molecule. The rate of 1st-order reactions depends only on the number of molecules present. The syntax of this kind of reaction is:

  $$r, \texttt{A} > \texttt{B + C + ...}$$

  A peculiar case of first-order reaction is when the right term of the chemical equation contains only the term NULL. This can be considered as an excretion reaction, where some chemical is released outside the chemical system. These reactions can be viewed as the opposite of 0-order reactions, at least as long as an uptaking systems can be thought as the opposite of an excreting system. In this case the syntax is:

  $$r, A > \texttt{NULL} + \ldots$$

- 2nd-order reactions. These reactions feature two species on the left side of the chemical equation. Since this kind of reaction takes place when one molecule of the first species collides with one molecule of the second species, the rate of 2nd-order reactions depends on the number of molecules of the first species, the number of molecules of the second species, and on the volume of the system.

2nd-order reactions are the most common reactions in the cellular scenario, also because a higher-order reaction can be described as a combinatorial collection of several consecutive 2nd-order reactions.

- Immediate reactions. They take place immediately after the verification of the condition represented by the left side of the equation. In other words, the stoichiometry noted at the left side of the equation is a logical condition: once it is verified, the immediate reaction takes place and yields the products indicated at the right side. These reactions allow for complex stoichiometries, where both the reagents species number and the number of molecules per species can be greater than two. We want to remark that immediate reactions are not to be considered higher-order reactions since they do not have a kinetic law, and they happen immediately. The general syntax for an immediate reaction is:

  $$-,\ \texttt{A + B +}\ldots > \texttt{C + D +}\ldots$$

  The initial dash sign declares that this is an immediate reaction, with no associated reaction rate.
  Immediate reactions can also be used to simulate the simultaneous excretion of several chemical species, once they have reached a given threshold number of molecules: to do so, the right side of the equation must contain only the "NULL" term.

The fourth block contains the number of molecules assigned to all the declared chemical species that are supplied to the system, along with the time at which they are supplied. When such time is set to zero, the statement denotes the initial number of molecules for the declared species.

The last two blocks are optional and concern with the eventual presence of control variables, and in particular:

The fifth block contains all the control variables used during the experiment, their name begins with a dollar sign. As for the species names, multiple variables can be written on a line, separated by a comma; e.g.: $\$k$, $\$y$, ...

The sixth block declares at what moment the control actions have to be executed and what is the final value of the controlled variable after the control action. To assign a value to a control variable at the beginning of the simulation, the statement is:

$$0,\ \$k,\ v_0$$

e.g., the statement: 0, $\$k$, 100 assigns the value 100 to the variable $\$k$ at the beginning of the simulation.
If one wants to change such value during the simulation, the syntax is:

$$1,\ t,\ \$k,\ v_t$$

e.g.: 1, 10, $\$k$, 0.1 will indicates that at 10 sec of simulated time $\$k$ changes its value to 0.1.

7.3 QDC's core

Given a metabolic system (and eventual actions performed on it) described in QDC input format, QDC parses it into C++ source files, compiles and simulates the time course of a biochemical system using our C++ language implementation of the Gillespie's direct method. As the input rates of the reactions must be the reaction-rate constant k_j, QDC uses Table 7.1 for converting them into the stochastic reaction constant c_j required for the simulation. This procedure has been designed

R_j	Units of k_j	c_j
$NULL \xrightarrow{k} \dots$	$Msec^{-1}$	kN_AV
$S_1 \xrightarrow{k} \dots$	sec^{-1}	k
$S_1 + S_2 \xrightarrow{k} \dots$	$M^{-1}sec^{-1}$	$kN_A^{-1}V^{-1}$
$2S_1 \xrightarrow{k} \dots$	$M^{-1}sec^{-1}$	$2kN_A^{-1}V^{-1}$

Table 7.1: Conversion table between stochastic and deterministic kinetics constants. N_A is the Avogadro's constant.

to specialize the source file on the given model, thus allowing to fully exploit the compiler optimizations. QDC turns out to be able to perform comparably or even better than most of its competitors (see the "Benchmark test" section for a detailed discussion on this point). Suitable scripts are provided to automate all these steps. QDC can be used in a command-line environment: in this way, it does not need any graphical resource, and can be run on all the most common operating systems. But also using a graphical interface as we show in the following.

7.3.1 QDC's output

QDC outputs three files: the first contains the time course of the number of molecules of each simulated metabolite; the second contains the counters of each metabolic reaction firing and the third contains the time course of the propensity for each reaction. These outputs allow the user to test whether possible artifacts (for instance, those induced by stiffness) have occurred. By exploiting the time course of the concentrations and the actual rate of reaction firing it is possible to perform a deeper investigation of the chemical system.

7.3.2 SBML Import/Export

The QDC package contains also two applications (called `import_sbml.py` and `export_sbml.py`) that provide the Import/Export from/to `SBML` (`www.sbml.org`)by means of the `libSBML` v.4.0 libraries. Of course, such I/E is limited to the expressions and the statements that both the languages (SBML and QDC's one) support. The SBML I/E can be managed also via the GUI.

7.3.3 QDC's GUI

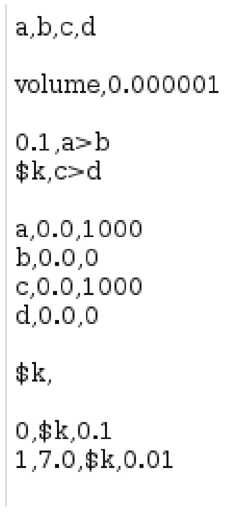
The QDC's GUI has been developed to allow the user to have an immediate visualization of the simulated system behavior. The GUI is basic and easy to use: it has been developed in Python v.2.6 language and uses the PyQt libraries to manage interface's elements. This choice confers QDC's GUI to be of good portability, as it has been tested on different Linux distributions (Fedora, Ubuntu, etc.) and on Mac OS X. The Fig. 7.2 shows the edit panel where one can directly write or paste the input file containing all the required blocks. Once completed the input, one can switch to the simulation tab where the simulation can be started by using the apt button; the next steps (compiling and running) are displayed once completed (Fig. 7.3). Lastly, the plot tab provides a direct access to the plotting of the output files. These are released as .CSV files, and the user can choose which one to plot in the plot window, by using the "load CSV" button. Then, the user can choose a single species to plot, by selecting it out of the list and clicking on "add plot" as shown in Fig. 7.4(a). The same goes for the reaction firing rates (Fig. 7.4(b)) and the propensity functions (Fig. 7.4(c)) files. That allows the user to see how the propensities change during the experiment.

7.4 Numerical experiments and Performance evaluation

Now we present some simple examples to see in practice the main features of QDC.

7.4.1 Some simple example

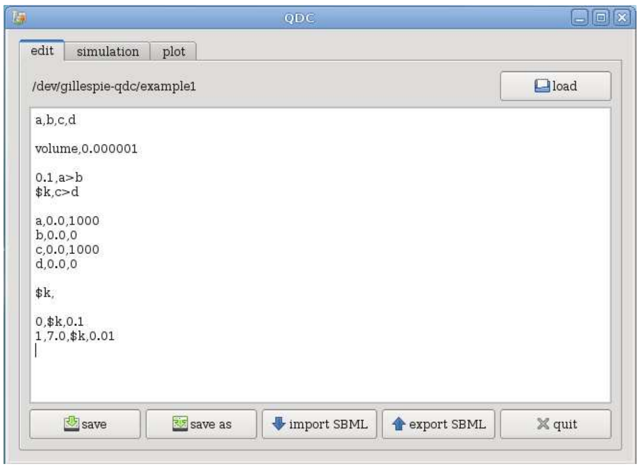
Fig. [7.2,7.4(c)] show an example of a QDC run of the biochemical system presented in Fig. 7.1. It consists of four chemical species interacting in two first-order reactions. The rate of the second reaction is controlled by the variable $k, which is set to 0.1 at the beginning of the experiment and switches to 0.01 after 7 seconds, as indicated in the second line of the block that assigns numerical values to symbolic variables. As it is clearly shown in Fig. 7.2, the reaction rate change modifies abruptly both the curves of the reagent species ("c") and product species ("d") of the controlled reaction, while it does not affect the other two species involved in a fixed rate



```
a,b,c,d

volume,0.000001

0.1,a>b
$k,c>d

a,0.0,1000
b,0.0,0
c,0.0,1000
d,0.0,0

$k,

0,$k,0.1
1,7.0,$k,0.01
```

Figure 7.1: An example of QDC input file: blocks are separated by blank lines.

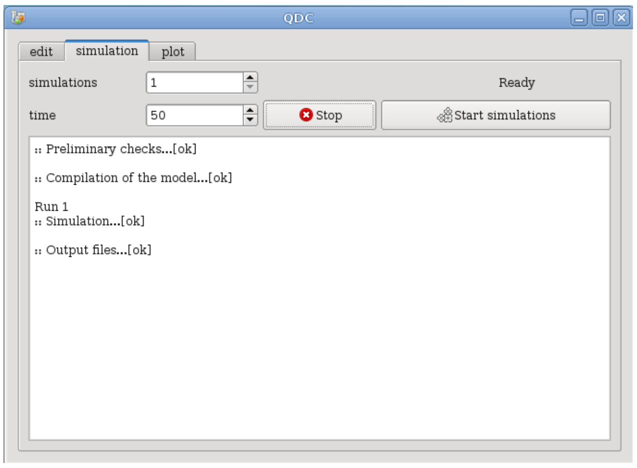
reaction. This is only one possible example out of many of a simple chemical system



```
/dev/gillespie-qdc/example1

a,b,c,d

volume,0.000001

0.1,a>b
$k,c>d

a,0.0,1000
b,0.0,0
c,0.0,1000
d,0.0,0

$k,

0,$k,0.1
1,7.0,$k,0.01
```

Figure 7.2: QDC GUI input form.

where a reaction is somehow controlled by an external factor. In Fig. 7.4, and in all the following one, the abscissa reports the number of computed samples during the simulation. As the sampling frequency is set at 0.1 Hz, 10 samples take 1 s of simulation time. In this case, the change in the reaction rate after 7 s is clearly visible as the 70th sampling point. A more sophisticated example, with no control actions, but with a very peculiar behavior by the system is shown in Figs. [7.5,7.7. We used QDC to face a stiff system described by Cao *et al.* [38], when they considered the problem of a generic enzymatic reaction. It is well known, in fact, that in enzymatic reactions the enzyme-substrate complex is usually much more likely to decay into its original constituents than to generate a product molecule. This is because the



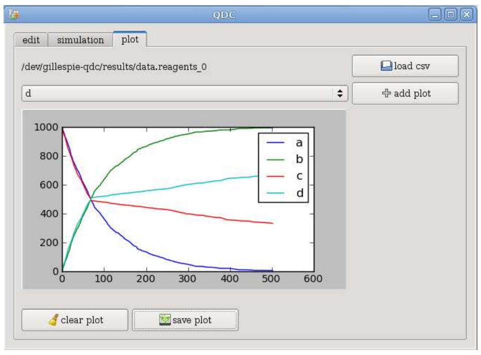
```
:: Preliminary checks...[ok]

:: Compilation of the model...[ok]

Run 1
:: Simulation...[ok]

:: Output files...[ok]
```

Figure 7.3: QDC GUI: monitoring the running status.

decay reaction has a much higher rate than that of product formation reaction. In this simple model, an enzyme S1 binds a chemical species S2 and generates a enzyme-substrate complex S3. This can decay back into S1+S2 or continue with the reaction giving a product S4 and the enzyme free form S1. The decay reaction rate is three orders of magnitude higher than the two others. Fig 7.5 shows the complete QDC input file for this system. In a stiff system the slow part of the system (the system subset with very low rate reactions) is called after a relatively long time, if compared with the fast part of the system. If only the time course of the number of molecules of each species is checked (see Fig. 7.6), it could erroneously be concluded that the system has reached an equilibrium state, even if this is not true at all. In the simulated stiff system, after a very short initial period (only one sampling period, 0.1 s), the species seem to have reached an equilibrium, as they possess a (quasi)-constant number of molecules. Two species are not visible here as their quantities exactly overlap those of the two displayed species. To be aware of this it is also necessary to inspect the actual firing rate of each reaction (II output file of QDC): this makes evident that reaction 3 does not take place at all during all the experiment duration (see Fig. 7.7). The effective reaction firing rate in the considered stiff system shows that there is one reaction (s3> s1 + s4) that has not yet taken place.

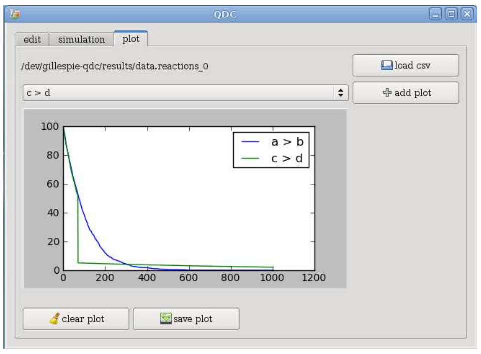
7.4.2 A benchmark test

To test the efficiency of our simulator we ran a benchmark test and we compare results with three widely used simulators, StochKit [98], Dizzy [135] and BetaWorkBench [58], which offer an implementation of Gillespie's direct method. We ran on the same computer several simulations of the same biochemical model (one strictly similar to that presented in Fig. 7.1, but without controlled variables). We varied the simulated time TIME, and we measured the execution time in seconds required


/dev/gillespie-qdc/results/data.reagents_0

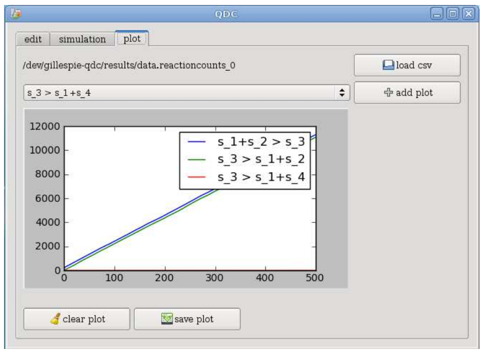
d

(a) Chemical species concentrations.


/dev/gillespie-qdc/results/data.reactions_0

c > d

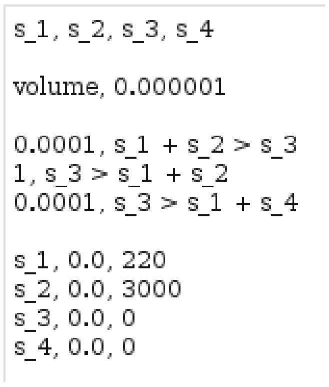
(b) Reaction firing rates.


/dev/gillespie-qdc/results/data.reactioncounts_0

s_3 > s_1+s_4

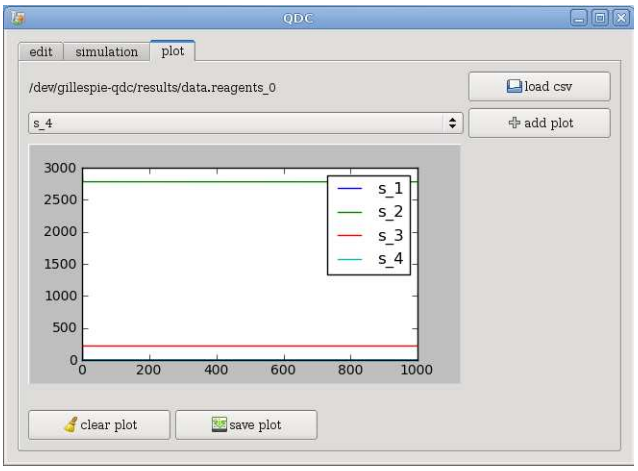
(c) Propensity functions.

Figure 7.4: QDC GUI output: plotting the time-course



```
s_1, s_2, s_3, s_4

volume, 0.000001

0.0001, s_1 + s_2 > s_3
1, s_3 > s_1 + s_2
0.0001, s_3 > s_1 + s_4

s_1, 0.0, 220
s_2, 0.0, 3000
s_3, 0.0, 0
s_4, 0.0, 0
```

Figure 7.5: An example of a stiff system input file.



/dev/gillespie-qdc/results/data.reagents_0

Figure 7.6: Plotting the time-course of the chemical species concentrations for the stiff system.

to perform the task. We plotted the execution time required by the simulators for each value TIME. In this curves we identifies the region where they are quasi-linear (when the simulation requires a number of operations significantly greater than those necessary to launch the program, but not so high as to require swapping). By using a linear regression, we determined the slope of the lines, which represents the coefficient linking the elapsed time TIME to the simulation time. Smaller slope corresponds to higher efficiency. Here we report these slope values for the simulators examined. StochKit $= 7.25\ 10^{-3}$, Dizzy $= 5.8\ 10^{-3}$, BetaWorkBench $= 4.09\ 10^{-3}$ and QDC$= 2.75\ 10^{-3}$. The values are (each within a very small error, less than two order of magnitude, as we have taken 100 measurements of the same point)

These tests do not represent a proper study on QDC's complexity. They represent only an indication that QDC's efficiency is comparable or even better than that of other freely available simulators. In other words, QDC has successfully conquered the goal to open the Gillespie's SSA based simulation of biochemical systems to new

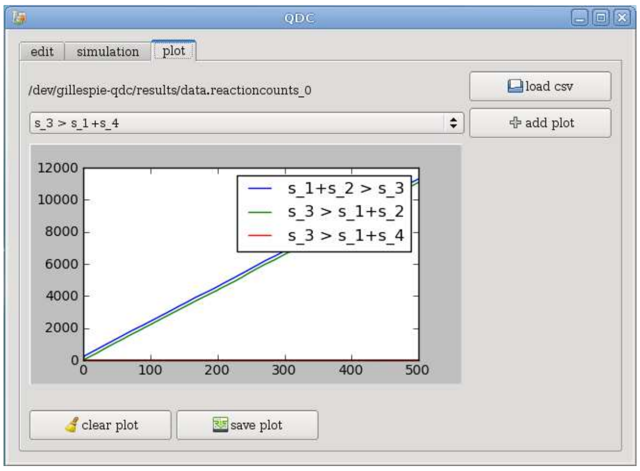


Figure 7.7: Plotting the time-course of the reaction firing rates for the stiff system.

complex features, without any loss of computational performances.

7.5 Conclusions

QDC (Quick Direct-method Controlled) is a simulator specifically designed to *in silico* reproduce wet-lab experiments performed on metabolic networks with several possible controls exerted on them by the operator. To execute a correct stochastic simulation of biochemical systems, QDC offers a very efficient implementation of the direct method version of the Gillespie's Stochastic Simulation Algorithm (SSA). It allows the user to simulate experimental controls in different ways; the user can: add or remove chemical species at a given time (even after the simulation has started); change the rate of a reaction at a given moment; describe reactions with complex stoichiometry, that take place once the stoichiometric condition is verified (here called *immediate* reactions). Moreover, even though QDC is not designed to manage compartments or membranes, it can simulate uptaking and excreting reactions through the usage of an operator called `NULL`. QDC is useful in particular for the simulation of signaling pathways, where the reaction parameters change upon specific triggering signals activated by the stimulus. The QDC syntax is designed specifically to be very close to the usual syntax of biochemistry, and it extends in a natural way the description of the action performed on the simulated system. Anyway, the import/export to/from SBML (Systems Biology Markup Language, www.sbml.org) level 2 is allowed, within the limits of the commonly supported features. QDC provides also a graphical interface designed by using Python language v.2.6, to ensure a good portability on the most common platforms. It is available as a GPL v.3 licence, and can be downloaded at `http://sourceforge.net/projects/gillespie-qdc`. The core is platform independent; while the GUI works on Linux or MacOSX.

Chapter 8

In silico testing of qualitative photoperception model

By realizing our QDC simulator we have been able to described the seven state model and to perform on it a large gamma of *in silico* experiments equivalent to those performed in vivo and reported in literature. This allowed us to explore the qualitative dynamic behavior of the proposed model and to assess if the model accounts for all experimental observations or not.

8.1 Data collection

Chemotaxis and phototaxis are very widely studied topics; the sensory mechanisms responsible are structurally and functionally conserved in many prokaryotes, and the understanding of their signaling pathway is of great interest for many research groups.

Most of the studies, however, are carried out on *E. coli* [6, 10, 9], and only parts of the discoveries about this gram negative enterobacterium are compatible with other organisms. *H. salinarum* has a similar pathway, but there are differences in the cytoplasmic molecules involved in the transduction chain: some are common to both *E. coli* and *H. salinarum*, others are found only in the first or in the second [4].

The literature about specifically *H. salinarum* was limited but very helpful in the understanding of both the basic and the complex responses in this archæon, and as many data as possible were retrieved from this source [5, 2, 8].

Much information used in this work about the reaction rates, peculiar aspects of the cell's behavior, and experimental evidences, however, comes also from unpublished results of the group of the CNR-IBF cooperating with ours.

Where lacking, constants were estimated in order to keep the system self-consistent.

8.2 The model

After an extensive research in literature, we chose the model introduced in Fig.6.2 that describes SRI's photocycle.

This model assumes transitions between seven spectroscopical species; four of them are responsible for the responses to orange light, while the other three mediate responses to blue light. The reason for this separation is in the experimental evidences that indicate that two different pathways are at work when the cells are stimulated with complex light stimuli.

In this proposed model, the signal transduction starts with four *signaling states*, two for each part of the cycle: two states are ultimately responsible for an increase in the reversal frequency, while the other two decrease it.

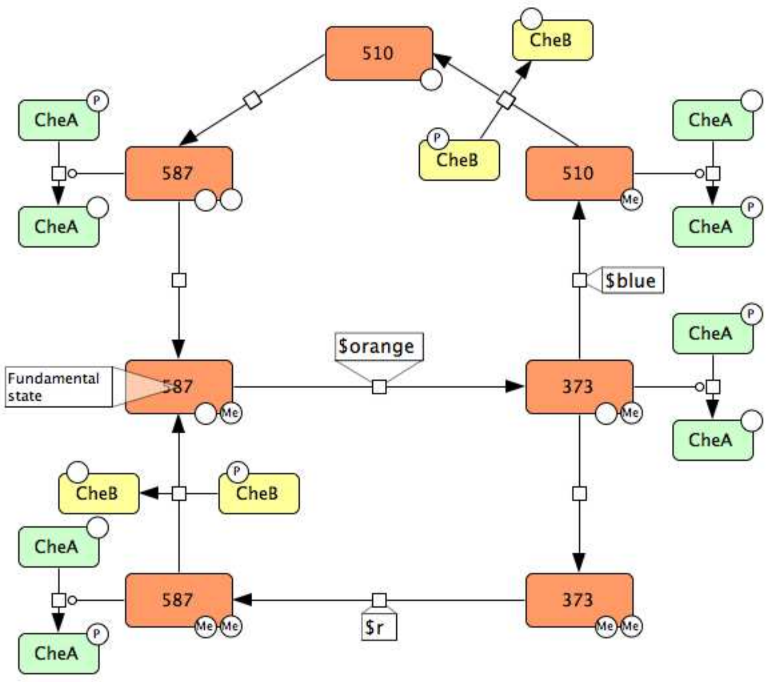


Figure 8.1: Our model for the photocycle

Starting from this structure, we set up a model formed by a photocycle of seven states, four of which are signaling states, whose role is - in our model - to phosphorylate or dephosphorylate the histidine kinase CheA[1] (Fig. 8.1). This role in nature is actually carried out by the transducer, that in our model is implicit in each state of the photocycle. The mechanisms of feedback and signaling therefore are included in the photocycle's activities.

[1]CheA is the autophosphorylating histidine kinase modulated by the signaling states; its activity includes the phosphorylation of CheB and CheY, resulting in its own dephosphorylation.

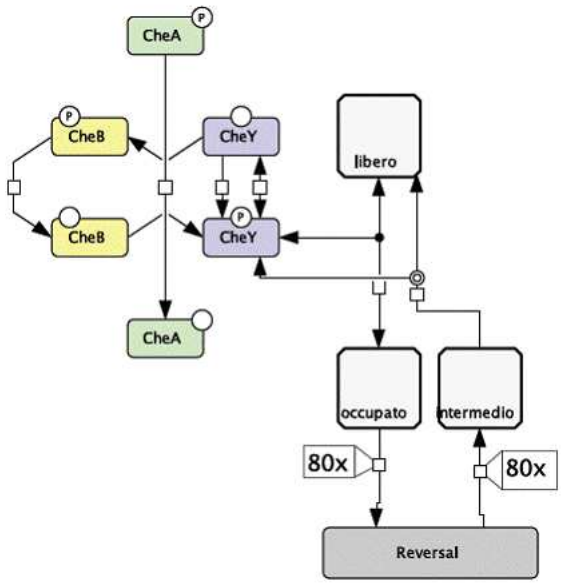


Figure 8.2: Signaling pathway from CheY-P to the flagellar motor

CheA is then responsible for the activation of CheB and CheY[2]. In this reaction, a phosphate is transferred from CheA to CheB or CheY. The loss of phosphate inactivates CheA, while the other molecule is activated.

Phospho-CheB brings the system toward its initial state by demethylating the activated states. In the process CheB-P return to its initial dephosphorylated state.

Phospho-CheY binds to the flagellar motor. We assumed that when a hundredth of the total concentration of CheY(-P) is bound to the motor, a reversal occurs (Fig. 8.2).

We assume for CheY-P a rate of autodephosphorylation and a much lower rate of autophosphorylation. Similarly, when bound to the flagellar motor, CheY-P can spontaneously detach from the motor before a reversal occurs, or can dephosphorylate and detach in the CheY form.

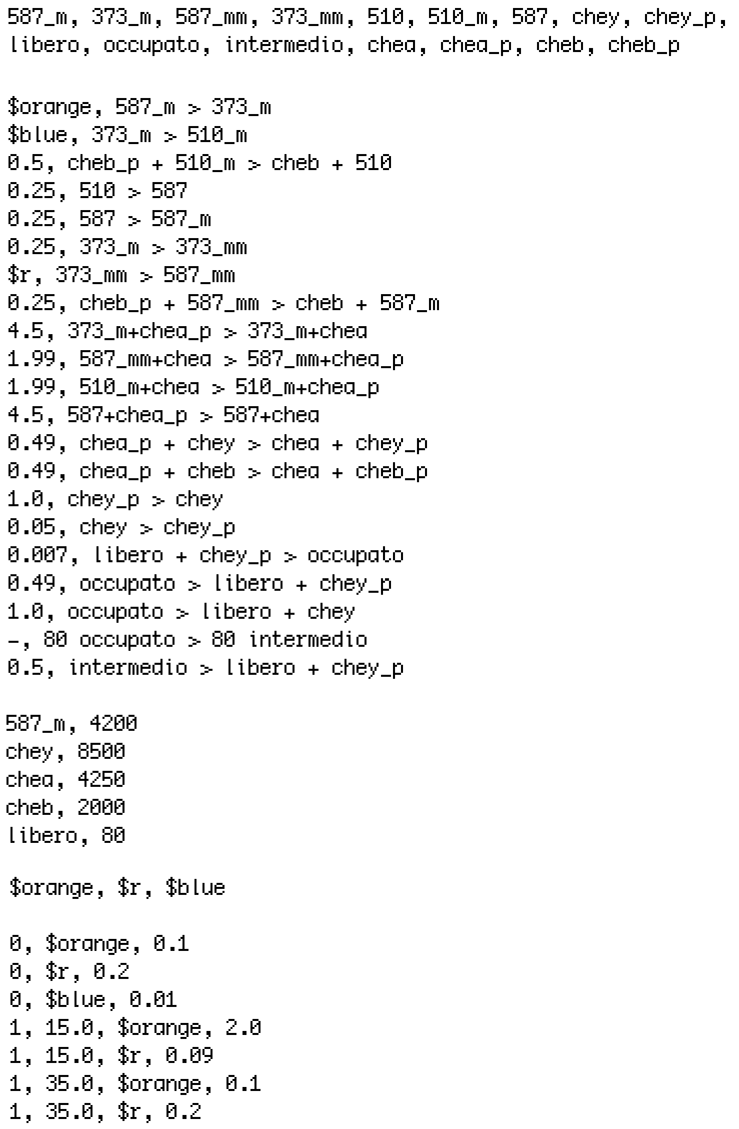
After a reversal, the motor loses the contact with CheY in a slow reaction that determines a temporary inhibition in the reversal occurrence. The motor releases CheY-P and reverts to its free form in two steps: first it converts in an intermediate state, then this state reconverts in the free motor state and CheY-P is released.

The rate of the passage from a state to another is set in some cases not by a constant but by a variable associated to the transition from light to dark and vice versa. Different variables account for the blue and orange lights signals. An

[2]CheY (in its phosphorylated form) is the cytoplasmic agent responsible for the signal transduction to the flagellar motor. Deletions in the *cheY* gene (as well as in *cheA*'s) results in a smooth swimming phenotype.

important case is the reaction from the fundamental state $SR587_m$ to $S373_m$. In absence of stimuli, in our model this reaction happens with a basal rate, enough to allow a photoisomerization in the dark. The same happens for the other reactions whose rates are set by variable. To allow photoisomerization in the dark is an important assumption for the model, and it accounts for both the random reversal frequency registered in absence of signals, and the response to blue light in the dark, which is an open debate as to whether it causes a repellent response or gives no response at all (see also chapter 8.3).

The QDC description of the model is summarized in Fig. 8.3.



```
587_m, 373_m, 587_mm, 373_mm, 510, 510_m, 587, chey, chey_p,
libero, occupato, intermedio, chea, chea_p, cheb, cheb_p

$orange, 587_m > 373_m
$blue, 373_m > 510_m
0.5, cheb_p + 510_m > cheb + 510
0.25, 510 > 587
0.25, 587 > 587_m
0.25, 373_m > 373_mm
$r, 373_mm > 587_mm
0.25, cheb_p + 587_mm > cheb + 587_m
4.5, 373_m+chea_p > 373_m+chea
1.99, 587_mm+chea > 587_mm+chea_p
1.99, 510_m+chea > 510_m+chea_p
4.5, 587+chea_p > 587+chea
0.49, chea_p + chey > chea + chey_p
0.49, chea_p + cheb > chea + cheb_p
1.0, chey_p > chey
0.05, chey > chey_p
0.007, libero + chey_p > occupato
0.49, occupato > libero + chey_p
1.0, occupato > libero + chey
-, 80 occupato > 80 intermedio
0.5, intermedio > libero + chey_p

587_m, 4200
chey, 8500
chea, 4250
cheb, 2000
libero, 80

$orange, $r, $blue

0, $orange, 0.1
0, $r, 0.2
0, $blue, 0.01
1, 15.0, $orange, 2.0
1, 15.0, $r, 0.09
1, 35.0, $orange, 0.1
1, 35.0, $r, 0.2
```

Figure 8.3: QDC input file for our model

8.3 Results and Discussions

Here we provide the experiments executed with our QDC simulator. We simulated the time evolution of 100 cells for 60 seconds. This required 100 independent simulations and time t=60. After each simulation we stored information about the final state reached, we set further operations on these data (e.g. counting the number of reversals) using some simple script written in the C and Perl programming languages.

We plotted the time each 0.001 seconds (axis x), the average concentration of CheY-P for the 100 cells (axis y in the left side of the plot) and the number of reversals happening during the simulation in the whole population (axis y in the right side of the plot). We also plotted the light intensities showing when light of a given spectrum is on or off.

The light condition tested are :

1. Dark (Absence of stimulation)
2. Long Lasting Orange
3. Blue Flash
4. Blue Flash on an orange background
5. Red and Blue Flash (white stimulus)
6. Sequence of orange pulses (Integration effect)
7. Orange step-down followed by a short orange pulse (Paradox effect)

The time required by our simulator vary very much according to the parameters of the model, the time spent for the simulation of our final model for 60 seconds of 100 cells is less than ten minutes.

8.3.1 Dark (Absence of stimulation)

In the absence of stimulation, *H. salinarum* performs spontaneous reversals with a frequency of one reversal every 5-50 s. We were able to reproduce this behavior, as shown in Fig. 8.5, where no stimulation has been supplied to the cells.

8.3.2 Long Lasting Orange

This experiment gives rise to a very complex behavior: at the orange light switch-on, there is a decrease in the reversals frequency. After a few seconds, there is an adaptation phase, during which the reversals frequency returns to its usual unperturbed value. At the orange light switch-off there is something like a step-down reaction, characterized by a strong increase of the reversals frequency. As it is shown in Fig. 8.6 the model show a qualitative agreement with the experimental data.(

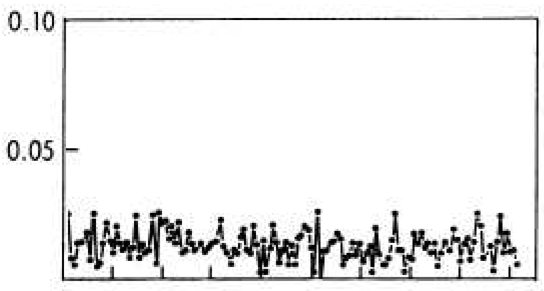


Figure 8.4: Experimental response of a dark condition, from [12]

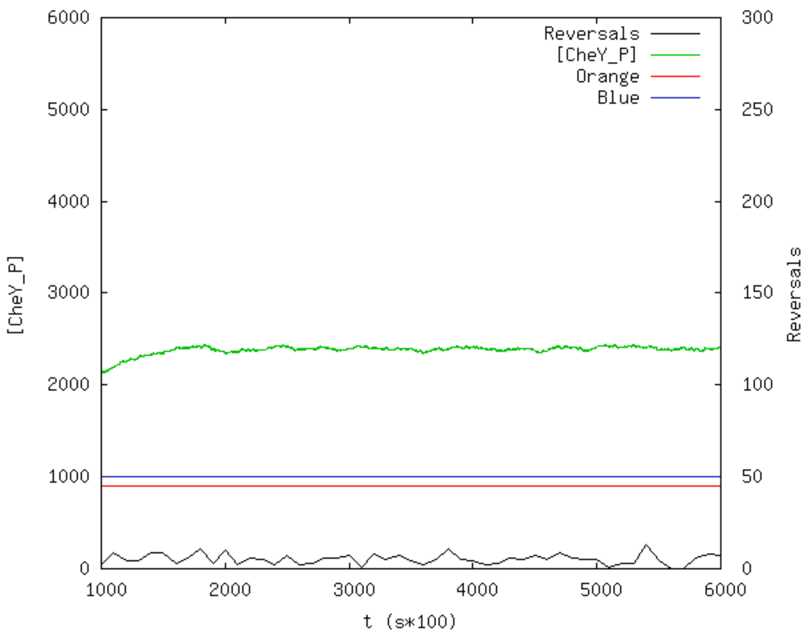
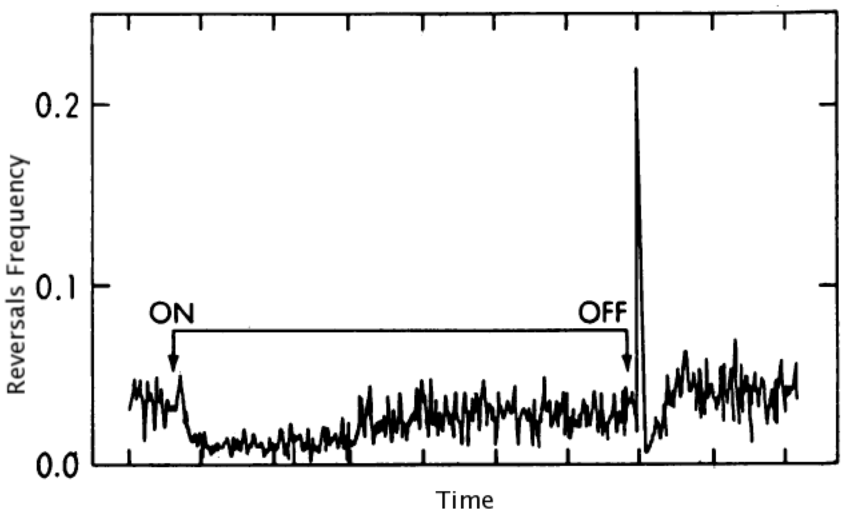


Figure 8.5: Simulation of a dark condition



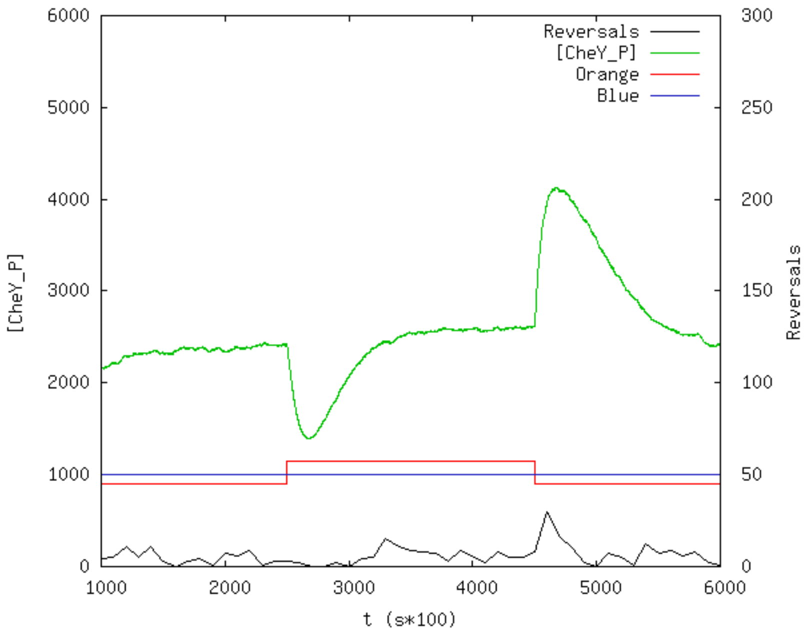


Figure 8.6: Responses to orange light. Top: experimental behavior, modified from [8]; bottom: our model's response.

8.3.3 Blue Flash

Experimentally, it is not clear whether the cells can or cannot respond to blue light in absence of other stimuli. The Hoff et al. model, for example, assumes a separate part of the photocycle for the responses to blue light, accessible only during an orange stimulation. On the other hand, some unpublished results suggest that this response is also possible when this is the only light given.

We therefore assumed for our model a photoisomerization in the dark, which allows the receptor to convert spontaneously also in the states involved in blue light responses. Our model, then, is able to respond to blue light stimuli starting from a dark condition (Fig. 8.7). The effect is a strong increase in the reversals frequency.

8.3.4 Blue Flash on an orange background

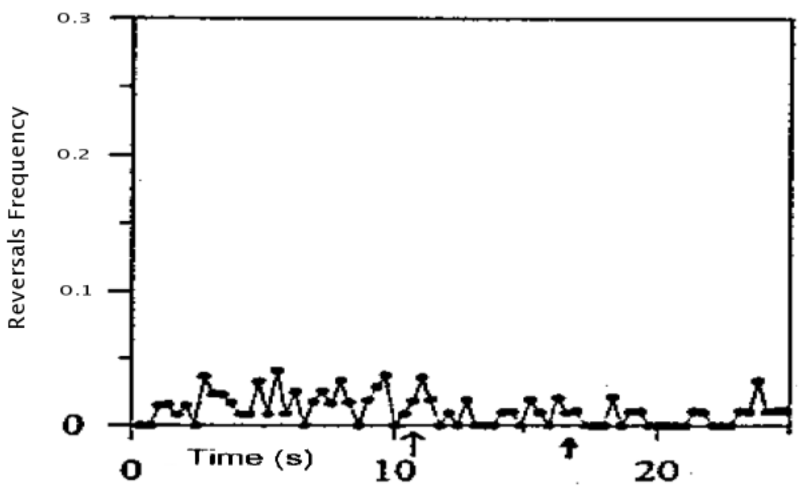
A similar condition tested is a blue flash on an orange light background. *In vivo* and *in silico* results show the same responses (Fig. 8.8):

- After the initial orange light step-up, the usual decrease in the reversals frequency is shown;
- 5-10 seconds later, the system adapts and the frequency returns to its basal value;
- A one-second long blue flash is given, and the cells respond with a rapid and strong increase in the reversals frequency;
- After the blue flash, the frequency returns to the pre-stimulus value;
- When the orange light is turned off, after a transient increase, the cells return to their normal basal frequency.

8.3.5 Red and Blue Flash (white stimulus)

As discussed in 6.3.1, white light, or a simultaneous orange and blue light stimulation, causes a strong repellent response similar to the one to blue light only. In Fig. 8.9, it is shown that our model's response to this stimulation agrees to the one experimentally registered:

- Immediately after the simultaneous orange and blue light step-up, the reversals frequency increases strongly;
- After the step-down, a rapid decrease in the frequency is registered and the system returns to its initial behavior.



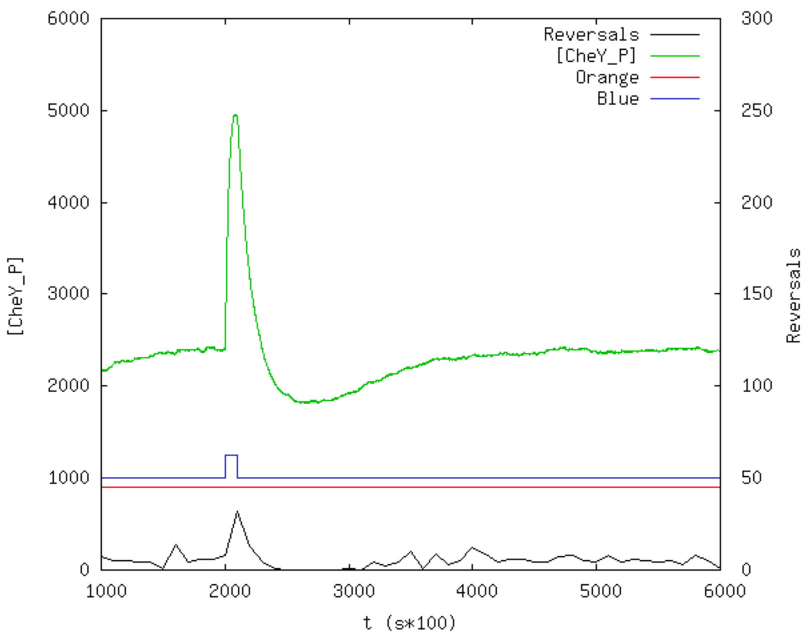
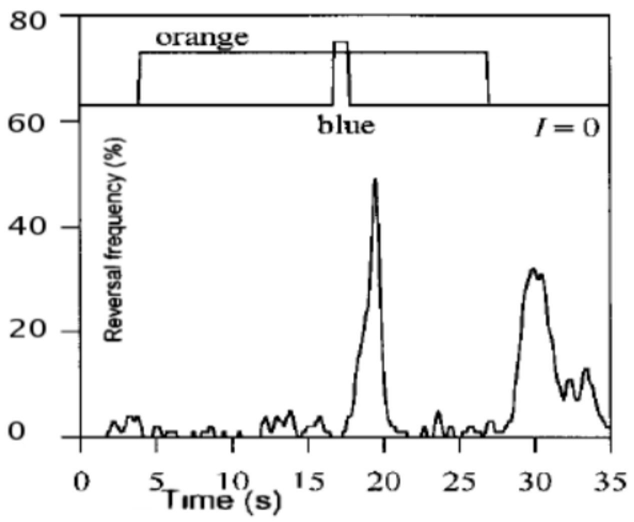


Figure 8.7: Our model's response to a blue light flash.



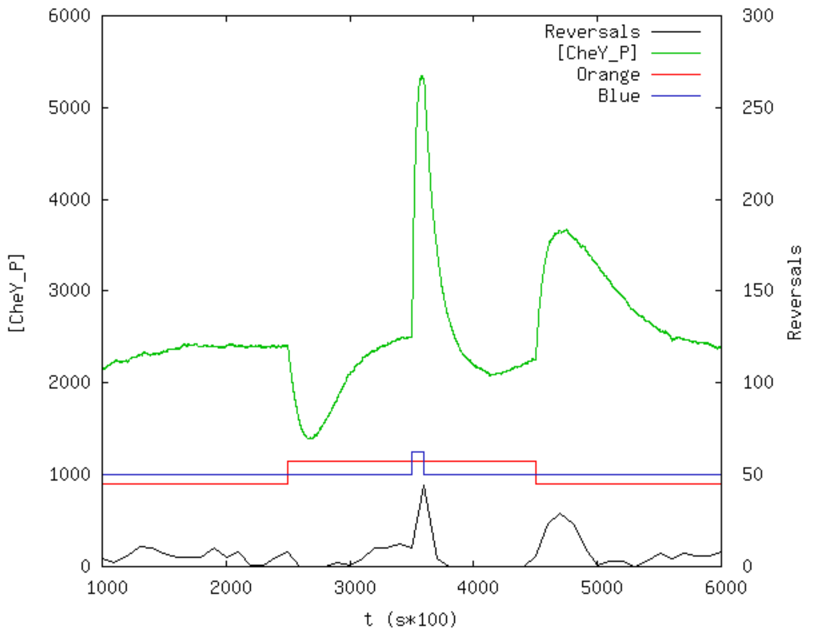
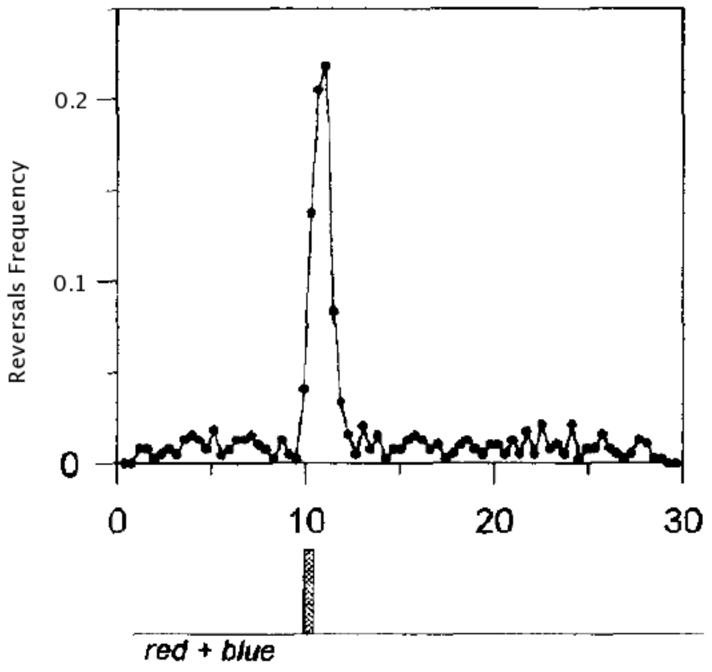


Figure 8.8: Blue flash on an orange background. Top: experimental response, from [2]; bottom: our model's response.



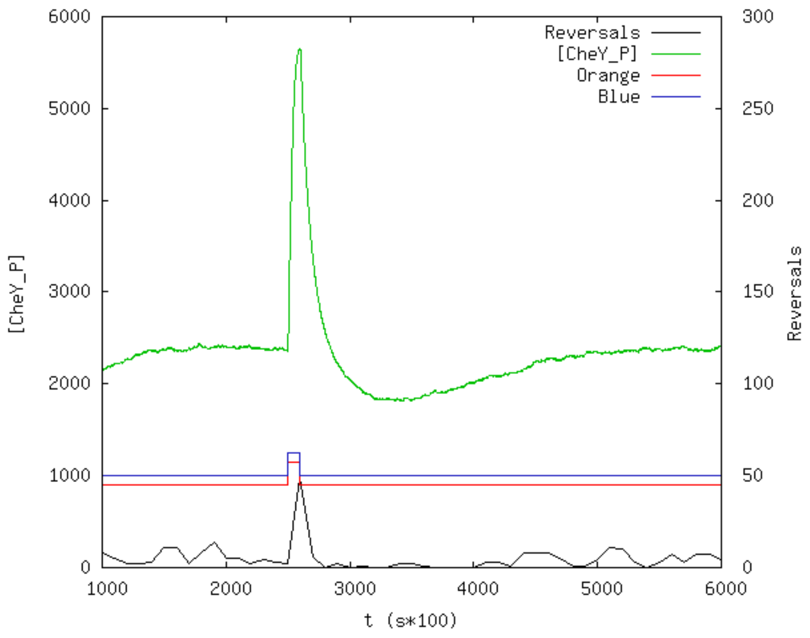


Figure 8.9: Top: red and blue flash response, modified from [5]; bottom: our result.

8.3.6 Sequence of orange pulses (Integration effect)

The integration effect is observed when a train of ineffective orange flashes is supplied to *H. salinarum.* If the delay between two consecutive flashes is shorter than a given threshold, after a certain number of flashes *H. salinarum* reacts as if had received a single flash with an intensity proportional to the sum of the intensities of the flash series. In other words, *H. salinarum* possesses the capability to integrate ineffective stimuli, and to store them in memory for a short time. Fig. 8.10 compares the *in silico* and *in vivo* behavior in response to a sequence of orange flashes:

- Each flash, if given alone, is not sufficient to determine a significative response;
- When a series of ten or more orange flashes is given in sequence, an integrate response is shown; in our model this is especially visible in the concentration values of CheY-P during the flashes;
- The result is that during this sequence of impulses, the reversals frequency decreases as if the pulses where a single, long stimulation.

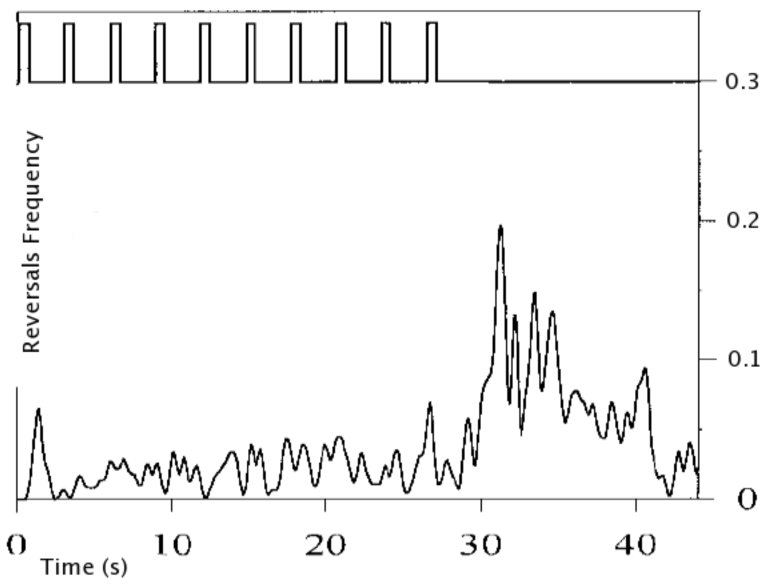
Both *in silico* and *in vivo*, this effect is visible only in specific conditions of duration and distance of the flashes; if the interval between a flash and the following one is too long, there is only a partial integration or no integration at all.

A difference between our result and the laboratory one is in the times of the stimulation: *in vivo*, the flashes are one second long and are given every three seconds; in our model flashes and intervals are ten times smaller. For longer intervals each flash determines its own response and no integrate behavior is shown, similarly to what happens *in vivo* if the time between a flash and the next is more than 4-5 seconds.

8.3.7 Orange step-down followed by a short orange pulse (Paradox effect)

The so-called "paradox effect" takes place when an orange flash is supplied about 12 seconds after the switch off of a long-lasting orange illumination. The paradox consists that this flashes generates an increase in the reversals frequency, rather than a decrease as usual. Such phenomenon suggests that the relaxation of the photomotile system after the long-lasting orange light exposure is a long process, during which there is a memory of the long-lasted previous illumination, so that the flash is perceived more similar to a switch off that to a switch on. Fig. 8.11 shows that in our model the second orange flash causes the usual decrease in the reversals frequency. In order to rule out possible critical dependence of this phenomenon on reaction kinetics, initial concentrations, etc., we performed an accurate and deep scanning analysis during which we varied:

1. The initial number of molecules for the whole model;



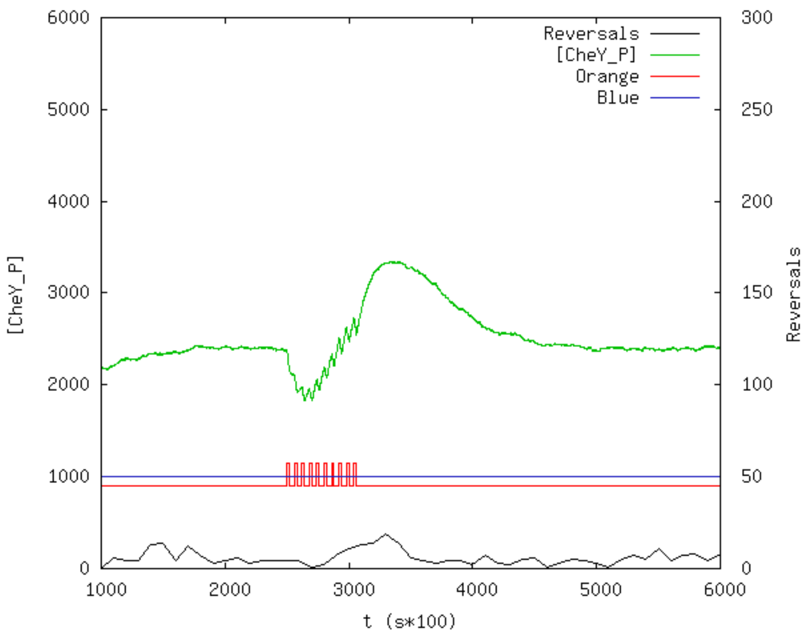


Figure 8.10: Top: stimulation with a sequence of orange pulses, modified from [3]; bottom: analogue stimulation in our model.

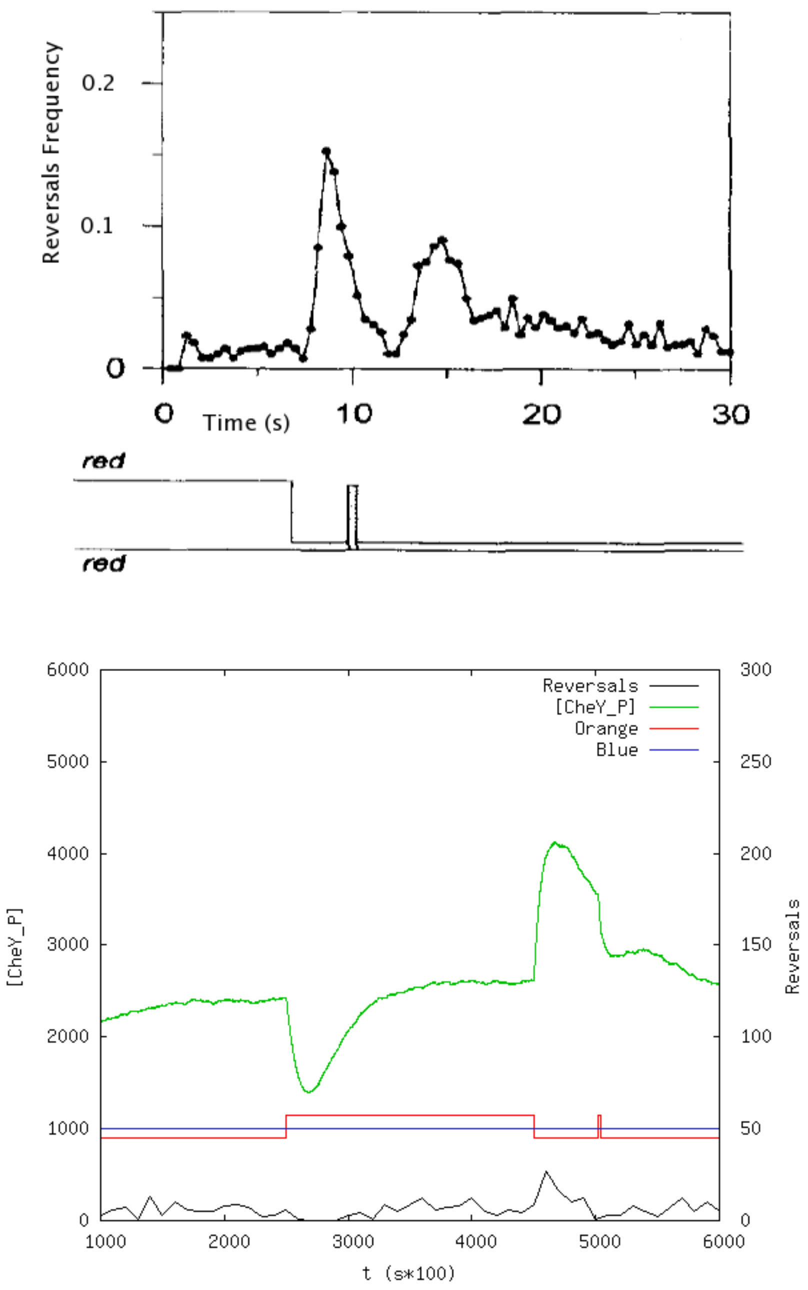


Figure 8.11: Top: *In vivo* response to an orange step-down followed by a short orange pulse, modified from [5]; bottom: the same stimulation in our model.

2. The flash duration;
3. The delay from the long lasting orange exposure and the flash supplying.

Our results showed impossible to reproduce the paradox effect in both the models. In our opinion this could be due to some missed component of the system the lifetime of which should be long enough to ensure a slow discharge of the photoreceptor transduction chain after the long lasting orange illumination. Of course, our models cannot give any indication about the nature of such missed component.

8.3.8 Conclusions

In this chapter we provided a quantitative stochastic model of the qualitative model proposed by Hoff *et al.* [4] to describe the SRI's photocycle. The aim of our study has been to verify what confirmed experimental behaviors of *Halobacterium salinarum* this accredited model is able to explain and eventually what it is not able to explain. To do this we build the quantitative stochastic model following meticulously the qualitative description of the model and we performed a very large gamma of *in silico* experiments varying different light and intensity stimuli. Then through the simulation of this model verify the results agreement between real wet-lab experiments taken from the literature and *in silico* results obtained by the simulations. To make it possible the comparison between real wet-lab experiments taken from the literature and the *in silico* results obtained by the simulations, we added the description of the signaling pathway from the photocycle's to the flagellar motor. The development of the quantitative model required to achieve kinetic data from the literature, when those data lack we took them by homology with *E. Coli.* To test the quality of the parameters we conducted a very meticulous robustness analysis of the model. In this analysis we spanned the value of the parameters by several order of magnitude and we check the results returned in correspondence of these parameters changes.

We were able to simulate many of the peculiar photoresponses of *Halobacterium salinarum* published in literature. Experimental conditions like dark, long lasting orange, blue flash, blue flash on an orange background compared with the experimental results are very similar and show at least a general agreement of the model to the simplest stimulations. More complex light patterns, like the sequence of orange pulses light stimulations showed also a qualitative agreement, whereas quantitatively our model differ from the times registered *in vivo*.

The *paradox effect*, on the other hand, seems impossible to show with the current model. This limit and the quantitative disagreement of the sequence of impulses suggests that our model lacks components that give it *memory*. This could be another state in the photocycle, leading to a more complex model than the one we adapted in this work, or something along the signaling pathway, which in this minimal model could be not adequately described, though seems to be sufficient for every other condition.

Concluding our quantitative analysis was able to reproduce many correct response observed experimentally, this make the approach a very promising validation tool. This confirmed that the qualitative model proposed by hoff explains many interesting observed behaviors. However, this model is not able to explain the paradox effect. Further works already in progress aim at finding the lacking parts needed to our system to show the paradox effect, as well as performing studies about the autocorrelation of the signals and expanding our system in order to have a more comprehensive understanding of the molecular mechanisms and interactions underlying the photoresponses in *Halobacterium salinarum*.

Part IV

Algorithmic Improvements to Stochastic Simulation

Chapter 9

Optimized Parallel FRM on Graphics Processing Units

In our research we also focused on the development of new efficient stochastic simulation methods. In this chapter we present a technological solution based on the multi-core `GPGPU`'s.

9.1 General-purpose GPU

Graphics Processing Units (GPUs) are multicore chips dedicated to and integrated into most of the modern video cards. As schematically illustrated by Fig. 9.1, a GPU devotes more transistors to data processing rather than data caching and flow control typical of CPU's. In this way they expect to be faster than CPUs in particular when the problem can be expressed as data-parallel computations[1] with high arithmetic intensity[2] [56]. Data-parallel processing maps data elements into parallel processing threads. Applications dealing with large data sets, such as arrays or matrices, can use a data-parallel programming model to speed up very much the computation. For example, in 3D rendering large sets of pixels and vertices are mapped and computed by parallel threads. For managing computations on GPU as a data-parallel computing device NVIDIA proposed the Compute Unified Device Architecture (CUDA) as high-level development environment [125]. In CUDA a GPU is viewed as a computing *device* capable of executing a very high number of threads in parallel. Any GPU co-operates like a coprocessor with the main CPU or *host*.

Giving an application in which a specific function is executed many times, but independently on different data, in this case the function can be isolated and executed on different portion of the data on the device by threads. To do that, such a

[1]The same program is executed on many data elements in parallel

[2]The ratio of arithmetic operations to memory operations must be high

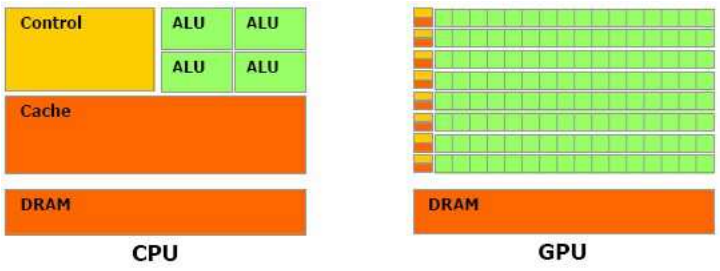


Figure 9.1: The GPU Devotes More Transistors to Data Processing

function is compiled to the instruction set of the device and the resulting program, called a *kernel*, is downloaded to the device.

Both host and device maintain their own DRAM, named *host memory* and *device memory*, respectively. One can copy data from one DRAM to the other through optimized API calls that utilize a specific tool named devices high-performance Direct Memory Access (DMA) engine.

Computational model

`CUDA` uses *stream computing* that is a computational model that represents a specific portion of data as a `stream`, and the program that specifies the operations on the streams as a kernel function. *Stream computing* enforces repetition of the same `kernel` on multiple independent data (`SIMD`). Each computation can be executed on the multiple processing units or `core`s. This means that `CUDA` represents a `GPU` as a matrix of computational cores capable of running `kernel`s. As illustrated in Fig. 9.2 each kernel is executed as a batch of threads organized like a grid of thread blocks.

Programming model

The `CUDA` programming interface consists of a minimal set of extensions to the `C` language and a runtime library. These extensions allow to the programmer to target `kernel` code. When a `kernel` is invoked, a number N of different `CUDA` threads execute it in a `SIMD` fashion. Logically, `kernel`s are executed as a grid of thread blocks. Threads within a *block* can cooperate by sharing data through the shared memory, and they can synchronize to coordinate memory accesses. The number of threads per block is restricted by the limited memory resources, so it cannot be arbitrarily large. Different blocks run independently, so the order of block execution is not guaranteed and communication between them is not available. Moreover, `NVIDIA` gives developers support on `kernel` execution configuration; programmers can specify the number of blocks and the number of threads for each block as arguments to the execution configuration. These are evaluated before the actual execution takes

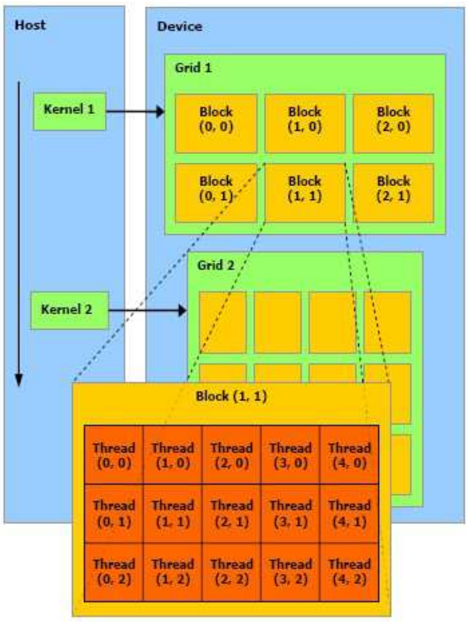


Figure 9.2: Thread Batching

place, `CUDA` also allows to perform read-modify-write atomic operations on one 32-bit or 64-bit words residing in local memory.

Memory model

The device is implemented as a set of SIMD multiprocessors. Each multiprocessor has on-chip memory of the four following types:

1. One set of local 32-bit registers per processor,
2. A parallel data cache or *shared memory* that is shared by all the processors and implements the shared memory space,
3. A read-only *constant* cache that is shared by all the processors and speeds up reads from the constant memory space, which is implemented as a read-only region of device memory,
4. A read-only *texture* cache that is shared by all the processors and speeds up reads from the texture memory space, which is implemented as a read-only region of device memory.

A thread has only access to the devices DRAM and the on-chip memory. It can read and write the *registers* and *local memory*. The threads in a block can read and write the *shared memory*, whereas the blocks in a grid can read and write the *global memory*, while read only from the *constant memory* and the *texture memory*. The

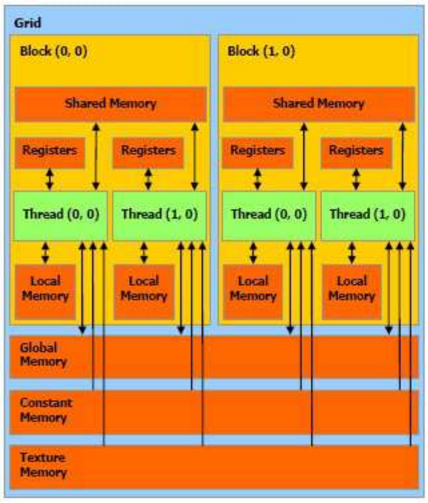


Figure 9.3: Memory Model

memory spaces model is illustrated in Fig. 9.3. So, the global, constant, and texture memory spaces can be read from or written to by the host and they are persistent across kernel launches by the same application. The global, constant, and texture memory spaces are optimized for different memory usages.

The `NVIDIA GPU`s can access both on-chip and off-chip memory. The on-chip is the fastest but also limited in dimension. For example, the shared memory capacity not exceeds `16KB`. The off-chip is one order of magnitude larger but two orders of magnitude slower than the former. `CUDA` provides non-blocking memory instructions to allow concurrent execution of master and `kernel` processes. For the reasons mentioned above, an effective on-chip memory and off-chip memory usage is a fundamental step for efficient computations in GPGPU's.

Compilation Toolchain

Source files for CUDA applications consist of a mixture of conventional C++ host code and GPU device functions. The CUDA compilation separates the device functions from the host code, it compiles device functions using proprietary NVIDIA compilers/assemblers. Then it compiles host code using any general purpose C/C++ compiler on the host platform. Afterwards, compilation embeds the compiled GPU functions as load images in the host object file. In the linking stage, specific CUDA runtime libraries are added for supporting remote SIMD procedure calling and for providing explicit GPU manipulation operations, such as, allocation of GPU memory buffers and host-GPU data transfer.

Interested reader can refer to [56] and the document at URL `http://sbel.wisc.edu/Courses/ME964/2008/Documents/nvccCompilerInfo.pdf`.

9.2 Sequential implementation

To take advantage of the GPGPU capabilities, an intensive use of their on-chip memory is necessary. This imposes any GPGPU programmer to consider seriously implementations with limited memory availability. Gillespie's DM requires to store N integers for the state $X(t)$ and M floating points or doubles for the propensity functions a_j. This can be a very good solution in a CPU based implementation but through some simple shrewdness we expect to propose a less space demanding implementation based on FRM.

The original version of FRM requires to store the state and at least M single or double precision variables for the propensity functions and M for all tentative times. The complexity in space of DM and FRM coincide to $\Theta(N + M)$. In our reformulation we generate the minimum tentative time incrementally. In other words, a propensity function and a random number are computed only when we want to compare the tentative time with the minimum among the times already computed in this minimum generation step. After the generation and the comparison a tentative time remains stored only if it is the minimum, otherwise it is removed. At the end of this iterative process the τ^m and j^m computed as minimum are in exact accordance with Eq. 5.3 and Eq. 5.4. This makes our algorithm equivalent to the original formulation of FRM. We summarize the main steps of our FRM re-definition in Alg. 4. Although the complexity in time of the algorithm remains essentially the

Algorithm 4 First Reaction Method Re-Formulation

```
while $t < TIME$ do
  $\tau^m \leftarrow \infty$
  $j^m \leftarrow -1$
  for j=1 to M do
    Compute $a_j(\mathbf{x})$
    Generates $r_j$ in $U(0,1)$
    Generate values for $\tau_j$ according to Eq. 5.2
    if $\tau_j < \tau^m$ then
      $\tau^m \leftarrow \tau_j$
      $j^m \leftarrow j$
    end if
  end for
  $t \leftarrow t + \tau^m$; $\mathbf{x} \leftarrow \mathbf{x} + \nu_{j^m}$;
  print $(t, j^m)$
end while
```

same of FRM, its complexity in space changes noticeably. In fact, the incremental computation requires to only store the system state $X(t)$, one a_j, one τ_j and the pair (τ^m, j^m). So the complexity in space now becomes $\theta(N)$. This reduces the order of required memory space of our re-definition with respect to both FRM and DM. We also introduce another memory saving optimization. This optimization regards the storing of the pair (t, j^m) instead of the more time consuming $(t, \mathbf{X})$, typical of the original version of FRM. In fact, the pair (t, j^m) describes the minimal information to determine the number of molecules over time. The motivation is that from one step to the next the state involves at most four molecules population changes. So storing the entire state `X` after each reaction generation substantially re-stores many unchanged state values. In literature, the file writing operation is often neglected, but we think that the number of writing of the state in the output file affects very much the execution time of FRM and all stochastic simulation methods.

Sequential Random Number Generation

As good statistical information can be achieved by independence of random numbers generated. In the sequential implementation of our algorithm we used a C language implementation of the well known *Mersenne-Twister* random number generator (`MT`) [116]. MT was designed to have a very long period of $2^{19937} - 1$, to generate negligible serial correlation between successive values, efficient usage of memory, good performance and portability.

9.3 Parallel implementation

Parallelism inside the simulation

In this section we describe how we implemented the sequential algorithm Alg. 4 in `CUDA`. At the beginning, the state $X(t)$ and the propensities values $a_j(x)$ must be transferred from the host to the device local memory. According to the data-parallel paradigm, we split M reactions into a number of portion H, and we assign each portion to a block, or task. Each task $h = 1, \cdots, H$ computes its local minimum τ_h^m according to the reactions assigned to its block, and then all tasks synchronize in order to find the global minimum τ^m. Then the resulting pair (τ^m, j^m) is sent back to the `CPU` for further processing and outputting operations. In order to obtain the best performance, our implementation hides the communication costs between device and host by overlapping `GPU` and `CPU` computations. The overlapping computation is possible using specific `CUDA` non-blocking communication operations between CPU and GPU and a double-buffering technique. In particular, double buffering consists in the following operations. While `CPU` reads from the buffer `B1`, the device updates information in the buffer `B2`. Then in the next step, the device writes into `B1`, while the `CPU` reads from `B2`. To minimize the size of data transferred, the kernel stores only the pair (τ^m, j^m). When CPU obtains a pair, it computes the next state and it stores this state into an output file. Alg. 4 summarizes the steps of this parallel version.

	Host process (CPU)	**Kernel program (GPU)**
1.	Initialization (seq. step 1)	
2.	Load $X(t)$ into GPU local memory	
3.	Spawn kernels using stream S2	
4.	Wait until buffer B2 is readable	Execute seq. step 2 on its block
5.		Write the pair (τ^m,j^m) back into B2
6.	Spawn kernels using S1	
7.	Read from B2 and Store (t, j^m)	Execute seq. step 2 on its block
8.	Wait until B1 is readable	Write the pair (τ^m, j^m) back into B1
9.	Read from B1 and Store (t, j^m)	
10.	if $t < t_f$ return to step 3	

To guarantee the maximum locality tasks communicate only after they terminate the local computation of the minimum. Whereas in order to maximize the usage of the limited shared memory, the number of threads per block is computed with the formula $C_X + P * (C_{a_j} + C_{r_j} + C_{\tau_j} + C_G) < 16 * 1024$. Here, C_X is the cost in byte to store the state $X(t)$ in the shared memory, P is the number of threads, C_{a_j}, C_{r_j} and C_{τ_j} are the cost in byte to store one propensity, one pseudo-random number and the tentative time in the shared memory, respectively. Then C_G is the shared memory used by the random number generator, and 16*1024 is the maximum size of the shared memory for one block.

Parallel Random Number Generation

In the parallel version, the multiple instances of `MT` should execute in parallel. However, the `MT` generator does not map very well onto the `GPGPU` paradigm, since it is hard to make a single twister state update in parallel among several execution threads [106]. Moreover, even when using "very different" initial state values, it is possible to have the emission of correlated sequences by each generator that shares identical parameters. For these reasons, we used an implementation of `MT` which runs multiple `MT` instances in parallel and uses a special off-line library for the dynamic creation of `MT` parameters [115]. The library accepts the 16-bit `thread id` as one of the inputs, and encodes this value into the `MT` parameters on a *per-thread* basis, so that every thread can update the twister independently, while still retaining good randomness of the final output.

9.4 Result and discussion

To test the performance of our parallel implementation we selected the `Heat shock Response` model (HSR) of E. Coli [104]. `HSR` describes the mechanism elaborated by E. Coli to respond to a fast temperature increment. The model involves 28 species participating in 61 chemical reactions [42]. Both sequential and parallel implementations of `FRM` run in a `NVIDIA GeForce 8600M GS` installed on a notebook

MacBook Pro with an Intel Core 2 Duo 2.4GHz, 4GB `RAM` running Windows XP. The `GeForce 8600M GS` has 16 stream processors, 1.2GHz engine clock speed, 512MB on-board memory. For this model we set a bunch of numerical experiments. For each experiment we fixed the number of steps of the simulation, we ran sequential and parallel implementations and we stored the execution time for one simulation. Fig. 9.4 shows the CPUTime obtained. While Fig. 9.5 summarizes speed-up. Our

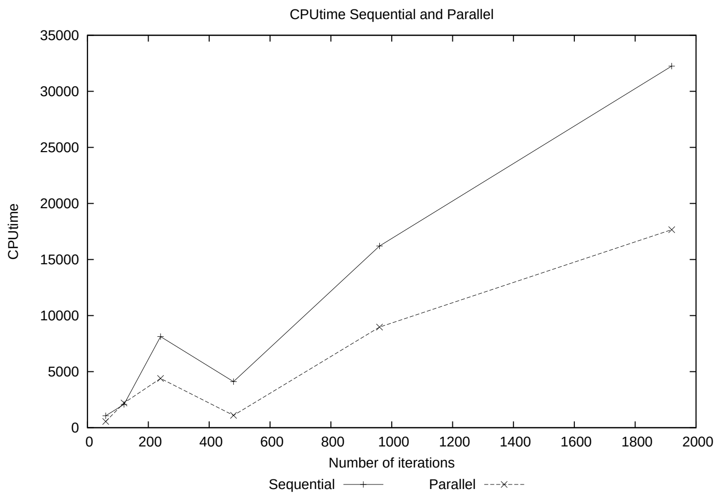


Figure 9.4: Comparison of CPUTimes (in milliseconds) of the sequential and the parallel version of `FRM` algorithm considering different number of simulation steps..

parallel achieves almost double speed-up compared with the sequential `FRM` version, and furthermore the performance scales with the number of simulation steps.

Although we obtained very efficient results for the biological model considered, in our opinion current `GPGPU`s programming environment is not yet mature to support general-purpose programming. `NVIDIA nvcc` has shown wrong behaviors during the tests. For example, some `C` language keywords are not recognized (e.g. `const`), and moreover emerged problems when translating non-trivial `C` code, especially using the `switch-case` control statements. We justify this issues as problems in the `NVIDIA`'s pre-processing step. We guess that the include files and macro invocations produce large files difficult to be managed by the compiler. This provokes the generation of executables that do not have the performance expected. To verify this hypothesis we performed a numerical experiment upon the signaling pathway of epidermal growth factor (EGF) receptor (EGFR) activated mitogen activated protein (MAP) kinase cascade [152]. This biological model involves 106 species and 296 reactions. During the compiling step the `NVIDIA` compiler failed the translation of the switch-case statement. This confirmed our hypothesis.

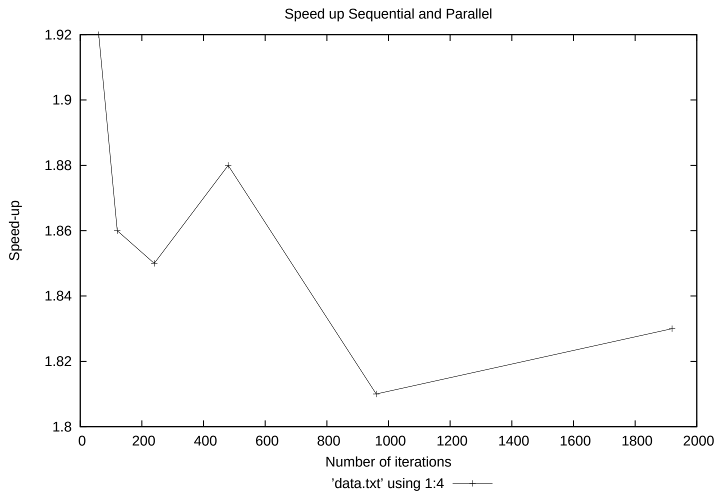


Figure 9.5: Speedups between sequential and parallel FRM considering different number of simulation steps.

9.5 Conclusion

In this chapter we have introduced a new sequential version of the FRM and a CUDA-based implementation to run it on modern GPGPUs. Our sequential re-formulation of FRM reduces space allocation by incrementally determining the minimum τ^m and by avoiding the storing of the state $\mathbf{X}(t)$ at each step. The parallel implementation on GPGPU halves the execution time, and we showed that this performance scales with the number of steps of the simulation. However, we highlighted that for our application the compiler provided some error augmenting the size of the model simulated. We conclude stating that in our opinion current GPGPUs programming environment is not yet mature to support a general-purpose programming. In particular, for our scientific application based on stochastic simulation. However, we are convinced that the future evolution of this very promising technology will resolve all issues mentioned, enabling new efficient possible parallel implementations.

Chapter 10

SSALeaping: Efficient Leap Condition Based DM Variant

To achieve higher simulation speed we shifted on approximated solutions based on the famous τ-leaping approach. The original τ-leaping method [73] substituted the notion of reaction time with the notion of leap. A leap is a time interval within which hopefully many reactions fire. In general given a time interval τ', there is no way to predict the reactions that will fire in τ' unless to run the simulation. However, the τ-leaping overcomes the problem ground on a condition of the reactions activity, called *Leap Condition*. The leap condition enables to estimate the number of occurrences of a given reaction into the considered leap as a sample value taken from a Poisson distribution, this step is known as *Leap Approximation*. The method advances from one state to the next, applying cumulatively all the reaction occurrences computed in the leap approximation step. So if each leap fires many reactions, substantial gain in simulation speed can be achieved [73].

Unfortunately, sometimes an uncontrolled application of the leap approximation can lead some population to become negative. To solve this problem some authors provided interesting solutions [163, 48, 39]. One of the most known and efficient τ-leaping method is the Modified τ-leaping (MTL) [39, 41]. This method uses a new user defined parameter to split the set of reactions of the system. The division is useful to separate and manage differently, reactions that potentially risk to overdraw some of its reactants, from the others. Moreover, MTL shifts from the Gillespie's DM to the τ-leaping and viceversa according to a condition that depends on a further user defined parameter. We will survey it in detail in the next section.

Our method is a new efficient stochastic simulation algorithm, called *SSAL*, which combines the advantages of Gillespie's Direct Method and of the τ-leaping. SSAL basically works as a standard DM but it verifies efficiently if the leap has to be interrupted or not. If it is the case, SSAL starts a new leap, otherwise, it computes a new pair (τ, j) reusing the same propensities and the sum of the preceding step. Moreover, the careful verification of the leap condition makes it impossible that some population becomes negative as we will prove later on. We also provide the asymptotic time complexity of DM, MTL and SSAL for further analysis.

10.1 Stochastic Simulation Algorithm: DM

As we already pointed out in Chap. 5, DM computes the following elementary steps. First DM computes all propensity functions $a_j(\mathbf{x})$, then it computes the sum a_0, and generating r_1 and r_2 in $U(0,1)$ it processes values for τ and j according to Eq. 5.6 and Eq. 5.7. Afterwards, DM updates the time course $t \leftarrow t + \tau$ and the state $\mathbf{x} \leftarrow \mathbf{x} + \nu_j$ storing $(t, \mathbf{x})$ into the output file. Here we want to investigate costly operations and complexity of DM.

For a single algorithmic step the main costs of DM are:

1. C_{a_j}, to compute M propensity functions a_j,
2. C_{a_0}, to sum M propensity a_j and obtain a_0,
3. C_{2r}, to generate the two uniformly distributed random numbers r_1 and r_2,
4. C_τ, to find the next occurring time τ,
5. C_j, to find the next reaction to fire,
6. C_{update}, to update the system state and the simulation time.

Assuming constants operations like: multiplication, division, sum, comparison, assignment and random number generation in a single step the bottleneck operations and their complexity are the following.

1. C_{a_j} of order of magnitude $\Theta(M)$,
2. C_{a_0} of order of magnitude $\Theta(M)$,
3. C_j of order of magnitude $O(M)$.

The bottleneck costs obtained are in accordance with the bottleneck costs identified in [42]. Taking into account the costs and their complexity the complexity in time of DM is reported in Eq. 10.1.

$$\begin{aligned} T_{DM}(M,N,n) &= (C_{a_j} + C_{a_0} + C_{2r} + C_\tau + C_j + C_{update})n \\ &= \Theta((M + M + 1 + 1 + M + 1)n) \\ &= \Theta(Mn), \end{aligned} \tag{10.1}$$

where n is the number of steps of the simulation.

10.2 Modified τ-leaping and Efficient τ-Formula

In Chap. 5 we introduced the general idea of the τ-leaping approach, as well as the main issues and solutions proposed. Here we pay more attention on the mathematical aspects that characterize a τ-leaping method, and we focus on the description of the Modified τ-leaping.
The idea of the τ-leaping is that the simulation can be divided into contiguous subintervals, or *leaps*. Substantial speed up can be achieved if many reactions can fire into a leap and if the leap computation can be done expeditiously [73]. The key constructs of the τ-leaping are the *Leap Condition* and the *Leap Approximation*. Suppose that the system is in state $\mathbf{x}$ at time t, the leap condition states the existence of a time value τ', such that, during the time interval $[t, t+\tau']$, every propensity function remains approximately constant to the value $a_j(\mathbf{x})$ at time t. The first leap condition formulation [73] required that, for every reaction R_j, the absolute fractional change $\Delta a_j(\mathbf{x})/a_j(\mathbf{x})$ during the leap never exceeded a *user-defined tolerance parameter* $\epsilon \ll 1$. Mathematically, it is written as follows.

$$| a_j(\mathbf{x}(t+\tau')) - a_j(\mathbf{x}(t)) | \leq \max\{\epsilon a_j(\mathbf{x}(t)), 1\} \quad j = 1, \cdots, M. \tag{10.2}$$

The amount of approximation depends by a *user-defined tolerance parameter* $\epsilon \ll 1$. Reducing the value of ϵ, the leap condition admits fewer changes in the propensity functions in a leap. Consequently, in a run the number of leap increases, and this affects the execution time. However, larger number of leap increase the accuracy of the results because the method recomputes more frequently propensity functions.

The *Leap Approximation*, instead, approximates the number of times a given reaction R_j fires during the leap as the Poisson distributed random variable $\mathcal{P}_j(a_j\tau')$. According to $\mathcal{P}_j(a_j\tau')$ a τ-leaping method generates a random sample k_j for each reaction R_j. Then the state of the system at time $t+\tau'$ results by the application of the formula $X(t+\tau') = \mathbf{x} + \sum_{j=1}^{M} \nu_j k_j$, where ν_j is the state change vector of a reaction R_j. As the larger τ', the larger values for $\mathcal{P}_j(a_j\tau')$ and k_j, it is therefore important estimating the largest τ' consistent with the leap condition. In literature the procedure to select the largest τ' is named *τ-selection procedure*[73, 74]. One of the most accurate, easier to implement and fast to execute has been recently proposed by Cao et. al in [41]. The underlying strategy of this procedure is to bound the relative change in molecular populations in such a way that the relative changes in the propensity functions are all bounded by the value ϵa_j. Cao et al. reformulated the leap condition definition as follows.

$$| \mathbf{x}_i(t+\tau') - \mathbf{x}_i(t) | \leq \max\{\epsilon_i \mathbf{x}_i(t), 1\} \qquad i \in I_{rs.} \tag{10.3}$$

In Eq. 10.3 I_{rs} denotes the set of indices of the species participating as reactant to at least one reaction, whereas, the values ϵ_i are selected to approximatively bound by ϵ the relative changes in all the propensity functions [41]. This means that the leap condition formulation in Eq. 10.3 implies that reviewed in Eq. 10.2. Unfortunately, using the above procedure, sometimes the selected τ' can induce too large changes and the population of some reactant with few molecules can become negative. The analysis of the phenomenon revealed negative populations arise for two main reasons. Being the Poisson distribution unbounded, a sample k_j generated in the leap approximation can exceed the maximum number of times that R_j can fire before consuming one of its reactants. Than since each propensity function change gets estimated separately, two reactions sharing a common reactant, acting together may overdraw that reactant population [163, 48, 39, 131]. To deal with negative populations, some methods substituted the Poisson distribution with a bounded one, as for instance a Binomial or a Multinomial distribution [131, 48, 131]. Some issues pointed out in [39] for the binomial τ-leaping methods have been resolved in [39] with the Modified τ-leaping (MTL). The Modified τ-leaping is one of the most known and efficient method to solve the negative population issue. So we focus on it surveying in detail the strategy used to solve the problem.

The Modified τ-leaping splits the reactions set into two groups: the critical and non critical. The critical group is composed by reactions that risk to consume some of their reactants. The non critical is composed by reactions that have low probability to consume the population of some of their reactants. To split the reactions MTL first computes the maximum number of times a reaction R_j can fire before exhausting one of its reactants, denoted with the quantity $L_j(\mathbf{x})$ in Eq.10.4.

$$L_j(\mathbf{x}) = \min_{i \in I_{rs}} \left[\frac{x_i}{\mid \nu_{ij} \mid} \right] \tag{10.4}$$

Then by means of a new user-defined parameter n_c, MTL puts a reaction R_j in the critical group J_c, if $L_j(\mathbf{x}) < n_c$ or in the non critical J_{nc}, otherwise. MTL treats these two groups in two different ways. The non critical reaction set J_{nc} serves to generate the largest τ' consistent with leap condition in Eq. 10.3 according to the Formula 10.5.

$$\tau' = \min_{i \in I_{rs}} \left\{ \frac{\max\{\epsilon x_i / g_i, 1\}}{\mid \sum_{j \in J_{nc}} \nu_{ij} a_j(\mathbf{x}) \mid}, \frac{\max\{\epsilon x_i / g_i, 1\}^2}{\sum_{j \in J_{nc}} \nu_{ij}^2 a_j(\mathbf{x})} \right\} \tag{10.5}$$

If many reactions can be fired in a leap, the non critical reactions are simulated by the original version of τ-leaping. However, if few occurrences can be fired, MTL temporarily switches to DM for $q = 100$ steps. The switch condition is the following.

$$\tau' \leq \frac{p}{a_0(\mathbf{x})} \tag{10.6}$$

We recall that $1/a_0(\mathbf{x})$ is the mean waiting time for the next firing reaction considering the propensity of all reactions, and p is a third user defined constant usually initialized to *ten*. The critical set J_c is managed differently. MTL imposes that at most one critical reaction can occur during the leap. This reduces drastically the probability to incur into negative populations. It first determines the occurrence of the next critical reaction τ'' and then it compares the times $t + \tau''$ and $t + \tau'$. If

$t+\tau'' < t+\tau'$ then MTL reduces the leap time assigning $\tau' = \tau''$, then it selects the next critical reaction j_c setting $k_{j_c} = 1$ and $k_j = 0$ for all $j \in J_c/j_c$. For all $j \in J_{nc}$ it generates the samples k_j according to $\mathcal{P}_j(a_j\tau')$. If $t+\tau'' > t+\tau'$ MTL sets $k_j = 0$ for all $j \in J_c$ and it generates $k_j = \mathcal{P}_j(a_j\tau')$ for all $j \in J_{nc}$.

Schematically, giving M reactions and kinetic constants, N species, one initial state $X(t_0)$ and a stop time TIME, four user-defined parameter n_c, ϵ, q and p, MTL performs the elementary steps in Algorithm 10. The procedure CompareTimes in Alg. 6 simply determines if the next critical reaction occurrence requires to reduce the leap or not. It also computes the leap approximation step. Below, we analyze the algorithmic time complexity of MTL. The costs that we identified are:

1. C_{a_j} and C_{a_0}, the same costs seen for DM,
2. C_L, involves both the costs for the computation of the M quantity L_j and to split the reaction set,
3. $C_{\tau'}$, the cost for τ-selection formula Eq. 10.5,
4. C_{DM}, the cost to execute q DM steps,
5. $C_{\tau''}$, the cost to compute the firing time of the next critical reaction,
6. $C_{\tau'<\tau''}$, the cost to compute the leap approximation in case no critical reactions fires during the leap,
7. $C_{\tau'\geq\tau''}$, the cost to compute the leap approximation in case the next critical reaction fires during the leap,
8. C_{neg} and C_{update}, respectively, the cost of the N checks to find eventual negative populations and the cost to apply the formula $\mathbf{x}_i \leftarrow \mathbf{x}_i + \sum_{j=1}^{M} \nu_{ij}k_j$ and $t \leftarrow t+\tau$.

The corresponding orders of magnitude associated with the costs are the following.

1. C_{a_j} and C_{a_0} are $\Theta(M)$.
2. C_L is $\Theta(M)$ because L_j must be computed M times, and for each reaction it performs at least one division and one comparison to decide if it is critical or not.
3. $C_{\tau'}$ is $\Theta(N+M)$. The τ-selection formula finds the minimum among N tentative leap times, that is one for each species in I_{rs}, this is $O(N)$. Then as the maximum number of multiplications of each propensity a_j in $\sum_{j\in J_{nc}} \nu_{ij}a_j(\mathbf{x})$ or $\sum_{j\in J_{nc}} \nu_{ij}^2 a_j(\mathbf{x})$ of Eq. 10.5 is at most equal to the number of the reactants of R_j, Eq. 10.5 needs at most $\Theta(M)$ propensity multiplications. This is $\Theta(M)$.
4. C_{DM} is $\Theta(Mq)$.

Algorithm 5 Modified τ-leaping

```
while $t < TIME$ do
  for j=1 to M do
    Compute $a_j(\mathbf{x})$
  end for
  Compute $a_0$
  for j=1 to M do
    Compute $L_j$ according to Eq. 10.4
    if $L_j < n_c$ then
      $J_c \leftarrow J_c \cup j$
    else
      $J_{nc} \leftarrow J_{nc} \cup j$
    end if
  end for
  if $J_{nc} \neq \{\}$ then
    Compute $\tau'$ according to Eq. 10.5
  else
    $\tau' \leftarrow 0$
  end if
  repeat
    if $\tau' < (p \cdot \frac{1}{a_0})$ then
      $temp \leftarrow t$
      Execute $q$ steps of DM
    else
      if $J_c \neq \{\}$ then
        Compute $\tau'' = \frac{1}{a0_c(\mathbf{x})} \ln\left(\frac{1}{r_1}\right)$
      else
        $\tau'' = 0$
      end if
      Execute CompareTimes in Alg. 6
      $temp \leftarrow t$
      $t \leftarrow t + \tau$
      $\mathbf{x}_i \leftarrow \mathbf{x}_i + \sum_{j=1}^{M} \nu_{ij} k_j$
      for j from 1 to N do
        if $\mathbf{x}_i < 0$ then
          $\mathbf{x}_i \leftarrow \mathbf{x}_i - \sum_{j=1}^{M} \nu_{ij} k_j$
          $t \leftarrow t - \tau$;
          $\tau' = \tau'/2$
        end if
      end for
    end if
  until $t = temp$
  print $(t, \mathbf{x})$
end while
```

Algorithm 6 CompareTimes

```
if $\tau' \leq \tau''$ then
  $\tau \leftarrow \tau^1$;
  for all $j \in J_c$ do
    $k_j \leftarrow 0$
  end for
  for all $j \in J_{nc}$ do
    $k_j \leftarrow \mathcal{P}_j(a_j\tau)$;
  end for
else
  $\tau \leftarrow \tau''$
  $j_c \leftarrow j'$ such that $\sum_{j' \in J_c} a_{j'}(\mathbf{x}) > r_2 a0_0 c$
  $k_{j_c} \leftarrow 1$
  for all $j \in J_c/j_c$ do
    $k_j \leftarrow 0$
  end for
  for all $j \in J_{nc}$ do
    $k_j \leftarrow \mathcal{P}_j(a_j\tau)$;
  end for
end if
```

5. $C_{\tau''}$ is $O(M)$ because for $\mid J_c \mid= M$ the firing time of the next critical reaction requires to sum at most M propensities for a_{0c}.

6. $C_{\tau'<\tau''}$ and $C_{\tau'\geq\tau''}$ are ΘM. They are mutually exclusive, and both compute M values k_j's. To simplify we assume that the Poisson random number generation is $O(1)$.

7. C_{neg} and C_{update} are $O(N)$ and $\Theta(M)$ because MTL checks at most N values of the state to find negative populations and it computes at most M unitary operations, one for each k_j, respectively.

In order to improve the readability of the time complexity in Eq. 10.7 and Eq. 10.8 we grouped together some of the preceding costs forming two groups: C_{TLEAP} and C_{TSTEP}.

$$C_{TLEAP} = C_{\tau''} + \max\{C_{\tau'<\tau''}, C_{\tau'\geq\tau''}\} + C_{neg} + C_{update} \tag{10.7}$$

$$C_{TSTEP} = C_{a_j} + C_{a_0} + C_L + C_{\tau'} \tag{10.8}$$

Now, let n', n'' and n''' the number of shifts to DM during the simulation, the number of leap and the number of reactions fired in all the shifts to DM, respectively. Considering the orders of magnitude introduced above and assuming that M and N are of the same magnitude, the time complexity of MTL is reported below in

Eq. 10.9.

$$\begin{aligned} T_{MTL} &= (C_{TSTEP} + C_{DM})n' + (C_{TSTEP} + C_{TLEAP})n'' \\ &= \Theta(Mn''' + Mn''). \end{aligned} \tag{10.9}$$

Eq. 10.9 highlights that the time complexity of MTL is mainly affected by the shifts to DM (Mn''') and by the computation of the leap (Mn''). This point out that the efficiency of MTL depends very much by the number of step and leap computed. In other words, by the value of the sum $n''' + n'$. Now, comparing the complexity of MTL and DM if the sum $n''' + n'$ is of order of magnitude of n, the time complexity for MTL is $\Theta(Mn)$. In this case, the complexity of MTL and DM coincide. Instead, if $n''' + n'$ is of order of magnitude $\frac{n}{M}$, it results that $T_{MTL}(M, N, n', n'') = \Theta(n) < T_{DM}(M, N, n)$ and this means that asymptotically MTL performs better than DM.

10.3 Our Proposal: SSALeaping (SSAL)

The preceding asymptotic analysis highlighted the substantial gain in simulation speed that can be achieved by MTL if each leap fires many reactions and few shifts occur (i.e. the n'' and n''' are small). Apart the optimistic case, it can happen that n'' and n''' are not so small. In other words, MTL can frequently shifts to DM, continuing to perform some of the extra operations identified with the costs C_L, $C_{\tau'}$, $C_{\tau' \geq \tau''}$, C_{neg} and C_{update}. When frequent shifts to DM occur, the burden introduced by these extra operations can slow down the performance. This can lead MTL to be slower than DM as well. In particular, when M and N are large. The minimum number of reactions to fire in a leap that guarantee a good speed up of MTL with respect to DM is fixed by the parameter p. However, this parameter is an heuristic and an arbitrary constant. To deal efficiently with the bad cases described above we propose a new method, that we call *SSALeaping* or SSAL for short. The idea of SSAL is very simple. It generates values for τ and j according to Eq. 5.6 and Eq. 5.7, then it updates the system state $X(t)$ according to the state change vector ν_j and it checks if the changes in some population break down the leap condition in Eq. 10.3. If it is the case, SSAL recomputes all propensities and a_0, otherwise, for the next generation of the values τ and j it reuses the same a_j and a_0. The extra cost paid for the verification of the leap condition is small compared to the extra costs seen for MTL. The verification also allows to build leap adaptively, and it makes impossible that some species population becomes negative. To do that the property used by SSAL is described in the next section.

10.4 No Negative Populations

We consider a reaction R_j and we solve the inequality $\mid a_j(\mathbf{x}+\nu_{j'}) - a_j(\mathbf{x}) \mid > \epsilon a_j(\mathbf{x})$, that is, if the leap condition for R_j is violated. To consider the most general case we identify the maximum state change that a propensity function $a_j(\mathbf{x})$ can undergo when a reaction $R_{j'}$ fires. Below we list all possible representative cases for R_j with their relative maximum change.

The first case we consider is when R_j is the unimolecular reaction of type $R_j : S_1 \xrightarrow{c_j} Product(s)$. For the unimolecular reaction the maximum change of the propensity $a_j(\mathbf{x})$ happens for $\nu_{1j'} = -2$.

$$\begin{aligned} \mid a_j(\mathbf{x}+\nu_{j'}) - a_j(\mathbf{x}) \mid &> \epsilon a_j(\mathbf{x}) \\ \mid (\mathbf{x}_1 - 2)c_j - \mathbf{x}_1 c_j \mid &> \epsilon \mathbf{x}_1 c_j \\ 2 &> \epsilon \mathbf{x}_1 \\ \frac{2}{\epsilon} &> \mathbf{x}_1 \end{aligned} \tag{10.10}$$

Here, the leap condition of R_j is violated when $\mathbf{x}_1 < 2/\epsilon$.

The second case we consider is for the bimolecular reaction of type $R_j : 2S_1 \xrightarrow{c_j}$ *Products*. In this case, the maximum change of the propensity for R_j happens for $\nu_{1j'} = -2$ and we have the following inequality.

$$\begin{aligned} \mid a_j(\mathbf{x}+\nu_{j'}) - a_j(\mathbf{x}) \mid &> \epsilon a_j(\mathbf{x}) \\ \mid \frac{(\mathbf{x}_1-2)(\mathbf{x}_1-3)}{2}c_j - \frac{\mathbf{x}_1(\mathbf{x}_1-1)}{2}c_j \mid &> \epsilon\frac{\mathbf{x}_1(\mathbf{x}_1-1)}{2}c_j \\ \mid (\mathbf{x}_1-2)(\mathbf{x}_1-3) - \mathbf{x}_1(\mathbf{x}_1-1) \mid &> \epsilon \mathbf{x}_1(\mathbf{x}_1-1) \\ \mid -4\mathbf{x}_1 + 6 \mid &> \epsilon \mathbf{x}_1(\mathbf{x}_1-1) \end{aligned}$$

For the absolute value $\mid -4\mathbf{x}_1 + 6 \mid$ we distinguish two cases. If $-4\mathbf{x}_1 + 6 \geq 0$ we resolve in the following way.

$$\begin{aligned} \mid -4\mathbf{x}_1 + 6 \mid &> \epsilon \mathbf{x}_1(\mathbf{x}_1-1) \\ -4\mathbf{x}_1 + 6 &> \epsilon \mathbf{x}_1(\mathbf{x}_1-1) \\ \epsilon \mathbf{x}_1^2 - (\epsilon - 4)\mathbf{x}_1 - 6 &< 0 \\ \mathbf{x}_1 &< 2. \end{aligned} \tag{10.11}$$

The leap condition of R_j can be violated when $\mathbf{x}_1 < 2$. Instead, if $-4\mathbf{x}_1 + 6 < 0$ we treat it as follows.

$$\begin{aligned} \mid -4\mathbf{x}_1 + 6 \mid &> \epsilon \mathbf{x}_1(\mathbf{x}_1-1) \\ 4\mathbf{x}_1 - 6 &> \epsilon \mathbf{x}_1(\mathbf{x}_1-1) \\ \epsilon \mathbf{x}_1^2 - (\epsilon + 4)\mathbf{x}_1 + 6 &< 0 \\ 2 \leq \mathbf{x}_1 &< \frac{(\epsilon+4) + \sqrt{((\epsilon+4)^2 - 24\epsilon)}}{(2\epsilon)} \end{aligned} \tag{10.12}$$

Finally, consider the reaction of type $R_j : S_1 + S_2 \xrightarrow{c_j}$ *Products*, we have two interesting state change cases: $\nu_{1j'} = -2, \nu_{2j'} = 0$ and $\nu_{1j'} = -1, \nu_{2j'} = -1$. In the first case the result is the following.

$$\begin{aligned} \mid a_j(\mathbf{x} + \nu_{j'}) - a_j(\mathbf{x}) \mid &> \epsilon a_j(\mathbf{x}) \\ \mid (\mathbf{x}_1 - 2)\mathbf{x}_2 c_j - \mathbf{x}_1\mathbf{x}_2 c_j \mid &> \epsilon \mathbf{x}_1\mathbf{x}_2 c_j \\ \mid -2\mathbf{x}_2 \mid &> \epsilon \mathbf{x}_1\mathbf{x}_2 \\ \frac{2}{\epsilon} &> \mathbf{x}_1. \end{aligned} \tag{10.13}$$

Instead, the second case results as follows.

$$\begin{aligned} \mid a_j(\mathbf{x} + \nu_{j'}) - a_j(\mathbf{x}) \mid &> \epsilon a_j(\mathbf{x}) \\ \mid (\mathbf{x}_1 - 1)(\mathbf{x}_2 - 1)c_j - \mathbf{x}_1\mathbf{x}_2 c_j \mid &> \epsilon \mathbf{x}_1\mathbf{x}_2 c_j \\ \mid -\mathbf{x}_1 - \mathbf{x}_2 + 1 \mid &> \epsilon \mathbf{x}_1\mathbf{x}_2 \end{aligned}$$

The absolute value $\mid -\mathbf{x}_1 - \mathbf{x}_2 + 1 \mid$ has two distinct cases. If $-\mathbf{x}_1 - \mathbf{x}_2 + 1 < 0$ we resolve in the following way.

$$\begin{aligned} \mid -\mathbf{x}_1 - \mathbf{x}_2 + 1 \mid &> \epsilon \mathbf{x}_1\mathbf{x}_2 \\ \mathbf{x}_1 + \mathbf{x}_2 - 1 &> \epsilon \mathbf{x}_1\mathbf{x}_2 \\ \epsilon \mathbf{x}_1\mathbf{x}_2 - \mathbf{x}_1 - \mathbf{x}_2 + 1 &< 0 \\ \mathbf{x}_1(\epsilon \mathbf{x}_2 - 1) &< \mathbf{x}_2 - 1 \\ \mathbf{x}_1 &< \frac{\mathbf{x}_2 - 1}{(\epsilon \mathbf{x}_2 - 1)} \end{aligned} \tag{10.14}$$

Instead, if $-\mathbf{x}_1 - \mathbf{x}_2 + 1 \geq 0$ we treat it as follows.

$$\begin{aligned} \mid -\mathbf{x}_1 - \mathbf{x}_2 + 1 \mid &> \epsilon \mathbf{x}_1\mathbf{x}_2 \\ -\mathbf{x}_1 - \mathbf{x}_2 + 1 &> \epsilon \mathbf{x}_1\mathbf{x}_2 \\ \mathbf{x}_1(\epsilon \mathbf{x}_2 + 1) - \mathbf{x}_1 &< -\mathbf{x}_2 - 1 \\ \mathbf{x}_1 &= 0 \\ \mathbf{x}_2 &= 0 \end{aligned} \tag{10.15}$$

Next we give one numerical example to show how to use the bounds defined before. Suppose to have $\epsilon = 0.03$, a unimolecular reaction $R_j : S_1 \xrightarrow{c_j} \mathit{Product}(s)$ and $\mathbf{x}_1 = 10$. Assume that a reaction $R_{j'}$ fires and $\nu_{1j'} = -1$. Than if we check the leap condition on R_j, we obtain the following result.

$$\begin{aligned} \mid a_j(\mathbf{x} + \nu_{j'}) - a_j(\mathbf{x}) \mid &\leq \epsilon a_j(\mathbf{x}) \\ \mid (\mathbf{x}_1 - 1)c_j - \mathbf{x}_1 c_j \mid &\leq \epsilon \mathbf{x}_1 c_j \\ \mid 9 - 10 \mid &\leq 0.03 * 10 \\ 1 &\leq 0.3 \end{aligned}$$

Now $1 \leq 0.3$ is false, so the leap condition for R_j is violated, but as we expect $\mathbf{x}_1 = 10 < 66 = 2/\epsilon$. Similar examples can be provided for any reaction. In summary, Formulas 10.10-10.15 give the thresholds under which any state change, involving a population $\mathbf{x}_i$, violates the leap condition of some reaction of which the species S_i is reactant. Nevertheless, the leap condition is violated frequently, when species with small populations are involved in fast reactions. In those cases, fast reactions occur frequently and their firing violates the leap condition many times because they change the population of species with few molecules. In literature exist some discussions about the worst case conditions for τ-leaping methods. Cao et al. [44, 46] stated that τ-leaping methods still have difficulties in effectively handling the situation when multiple time and population scale coexist, particularly when a species with a small population is involved in a fast reaction. Harris et al. [85] state that small reaction subnetworks (e.g. reversible reactions) that have small populations and large rate constants are the main bottlenecks for explicit leaping algorithms. In summary, small numbers and stiffness are considered the conditions causing worst cases. This broadly accepted conclusions are undoubtedly true, but the Formulas 10.10-10.15 tell us more. They confirm numerically the above discussions providing also numerical thresholds under which a population lays into what is often named *small number* population.

10.5 Optimizations and Comparisons

SSAL uses some algorithmic optimization that we summarize below. The first is on the leap condition. We reformulate the leap condition by substituting $\epsilon_i x_i$ to value $\max\{\epsilon_i x_i, 1\}$ in Formula 10.3 yielding

$$\mid \mathbf{x}_i(t+\tau') - \mathbf{x}_i(t) \mid \leq \epsilon_i \mathbf{x}_i(t) \qquad i \in I_{rs.} \tag{10.16}$$

Here, I_{rs} is the set of species of the system that act as reactant at least in one reaction. Additionally, we introduce the constraint that if the leap condition of a given reaction is violated after the firing of a reaction $R_{j'}$, SSAL aborts the current leap maintaining $R_{j'}$ the last fired reaction. In this way, any leap fires at least one reaction. This makes Formulas 10.3 and 10.16 equivalent.
The second optimization increases efficiency in the selection of the reaction to fire. This optimization is the core of the *Logarithm Direct Method* (LDM) [181]. LDM accumulates the partial sums of the propensities and it stores them in an array A. The ordered sequence of partial sums enablemakes it possible a binary search that finds the position j such that subtotal satisfies $A[j] < a_0 r_2 < A[j+1]$.

Our algorithm depicted in Alg. 9 takes as inputs M reactions and kinetic constants, N species, one initial state $X(t_0)$, a stop time TIME and a tolerance parameter ϵ. Given a reaction R_j, we define the set $\mathit{Reactants}(R_j)$ as the indices $i \in \{1, \cdots, N\}$ such that $\nu_{ij} < 0$ and the set $\mathit{Products}(R_j)$ as the indices $i \in \{1, \cdots, N\}$ such that $\nu_{ij} > 0$. We also define the set $((\mathit{Reactants}(R_j) \cup \mathit{Products}(R_j)) \cap I_{rs})$, that is the set of reactants or products species of the reaction R_j that are also reactants of at least one reaction.

Algorithm 7 SSALeaping

```
while $t < TIME$ do
    Compute $a_1(\mathbf{x})$; $A[0] \leftarrow a_1(\mathbf{x})$
    for j=2 to M do
        Compute $a_j(\mathbf{x})$
        $A[j] \leftarrow A[j-1] + a_j(\mathbf{x})$
    end for
    $a_0 \leftarrow A[M-1]$
    Store a copy of $\mathbf{x}(t)$
    $OK \leftarrow true$
    $\tau' \leftarrow 0$
    while $t + \tau' < TIME$ and $OK = true$ do
        Generate $r_1$ and $r_2$ in $U(0,1)$ and generate values for $\tau$ according to Eq. 5.6.
        Through binary search find $j$ according to Eq. 5.7.
        $\mathbf{x}(t + \tau' + \tau) \leftarrow \mathbf{x}(t + \tau') + \nu_j$;
        $\tau' \leftarrow \tau' + \tau$;
        for all $i \in ((Reactants(R_j) \cup Products(R_j)) \cap I_{rs})$ do
            if $\mid \mathbf{x}_i(t + \tau') - \mathbf{x}_i(t) \mid > \epsilon_i \mathbf{x}_i(t)$ then
                $OK \leftarrow false$
            end if
        end for
    end while
    $t \leftarrow t + \tau'$
    print $(t, \mathbf{x})$
end while
```

For a single step the costs are:

1. C_{a_j}, C_{a_0}, C_{2r}, C_τ,C_{update}, the same of DM
2. C_{copy}, the cost to make a copy of the state $X(t)$.
3. C_j, the cost for the binary search,
4. C_{Leap}, the cost to verify the leap condition in Formula 10.16.

The costs have the following order of magnitude.

1. C_{a_j}, C_{a_0}, C_{2r}, C_τ,C_{update} are the same as for DM.
2. C_{copy} is $\Theta(N)$ because the state has N elements.
3. C_j is $\Theta(\log_2 M)$.
4. C_{Leap} is $O(1)$ because checking the leap condition involves few arithmetic operations for each reactant and product of the fired reaction. Each reaction involves at most four species.

Note that for SSAL a leap is a time interval between two consecutive violations of the leap condition. A leap can involve the firing of one or more reactions. However, in those cases in which the reactions fired in a leap are more than one, SSAL pays the costs C_{2r}, C_τ, C_j, C_{update} and C_{Leap} for each selected reaction, and the costs C_{a_j}, C_{a_0} and C_{copy} for each leap. If we denote the number of steps with n and the number of leap with k, we consider the costs and the complexity associated the time complexity of SSAL is written in Eq. 10.17.

$$\begin{aligned} T_{SSAL}(M,N,n) &= (C_{a_j} + C_{a_0} + C_{copy})k + (C_{2r} + C_\tau + C_j + C_{update} + C_{Leap})n \\ &= \Theta((M + M + N)k + (log(M) + 1)n) \\ &= \Theta(Mk + \log(M)n). \end{aligned} \tag{10.17}$$

Note that two of the three bottleneck costs of DM (e.g. C_{a_j}, C_{a_0}) now have to be computed only for each leap. Whereas the third bottleneck cost C_j takes only logarithm time. If we compare the complexity of SSAL with the complexity given for DM and MTL we obtain the following results. In the worst case $k = \Theta(n)$, the complexity of SSAL and DM coincide. In the same way the complexity of SSAL and MTL coincide if we consider $n''' + n'' = \Theta(n)$. Without loss of generality, supposing that k and n'' are of the same order of magnitude. Asymptotically SSAL performs better than MTL if it holds when $n''' > \frac{\log(M)n}{M}$. Whereas MTL performs better than SSAL otherwise. In other words, SSAL performs better than MTL if the number of reactions fired by MTL when it shifts to DM exceeds the bound $\frac{\log(M)n}{M}$.

10.6 Numerical Experiments

The asymptotic analysis estimates theoretical information about efficiency, costly operations and relations among those operations and simulation parameters. However, to give a more pragmatic comparison of SSAL, MTL and DM we investigated how efficiency changes in practice. We provided experimental tests that consider different model parameters taken from realistic biological models. Our tests are made upon the Decaying-Dimerizing, Map Kinase Cascade and LacZ/LacY models, which span from four up to hundreds of reactions. In the LacZ/LacY model the cell volume is assumed to grow in time. The population of two species are randomly determined from two Normal random variables, and the mean values of these variables grow together with the volume of the cell. To the best of our knowledge, existing toolkits that implement MTL (i.e. Stochkit), even though extensible, they do not allow to specify those model features yet. For this reason, we realized our C language implementations of DM, MTL and SSAL and we used those implementations to simulate LacZ/LacY and the others models. All experiments run on a WINDOWS XP personal computer with a 3.0 GHz CPU and 1 Gbyte memory. For each experiment, we collected the final states of a selected species taken from 1000 independent simulations. We estimated the accuracy of the results by computing first the histogram and the Kolmogorov distances [47] between 1000 samples of DM and MTL or SSAL. Then we compared those distance values with the so called *self distance* [47], interpreting the comparison as follows. The closer to the self distance a distance value is, the more accurate the method who has generated that samples will be. We performed the mean and variance of the histogram and Kolmogorov self distances according to the formulas given in [47]. The mean and variance for the histogram self distance are 0.079 and 0.49 respectively. Whereas for the Kolmogorov self distance are 0.0389 and 0.00136 respectively. Apart this in order to cover the most large possible range of cases and parameters, we repeated experiments considering different ϵ values. For each test we have taken the values for: (ϵ), CPU Time ($CPUTimeSSAL/MTL$), number of reactions fired per leap for SSAL and for MTL (m'/m''), histogram distance ($HDSSAL/HDMTL$) and Kolmogorov distance ($KDSSAL/KDMTL$). Whereas, only for MTL, we consider: (ϵ), the number of MTL shifts to DM (n'), the number of leap and reaction fired ($n''' + n''$), the number of executions of the branch ($\tau' < \tau''$), the number of executions of the branch ($\tau'' < \tau'$).

10.7 Decaying-Dimerizing Model

This first test model is taken from [74]. It consists of three species S_1, S_2 and S_3 (N=3) and four reactions (M=4). A monomer S_1 reversibly dimerises to an unstable form S_2, which can convert to a stable form S_3. We simulate the model using the following stochastic coefficients: $c_1 = 1.0$, $c_2 = 0.002$, $c_3 = 0.5$ and

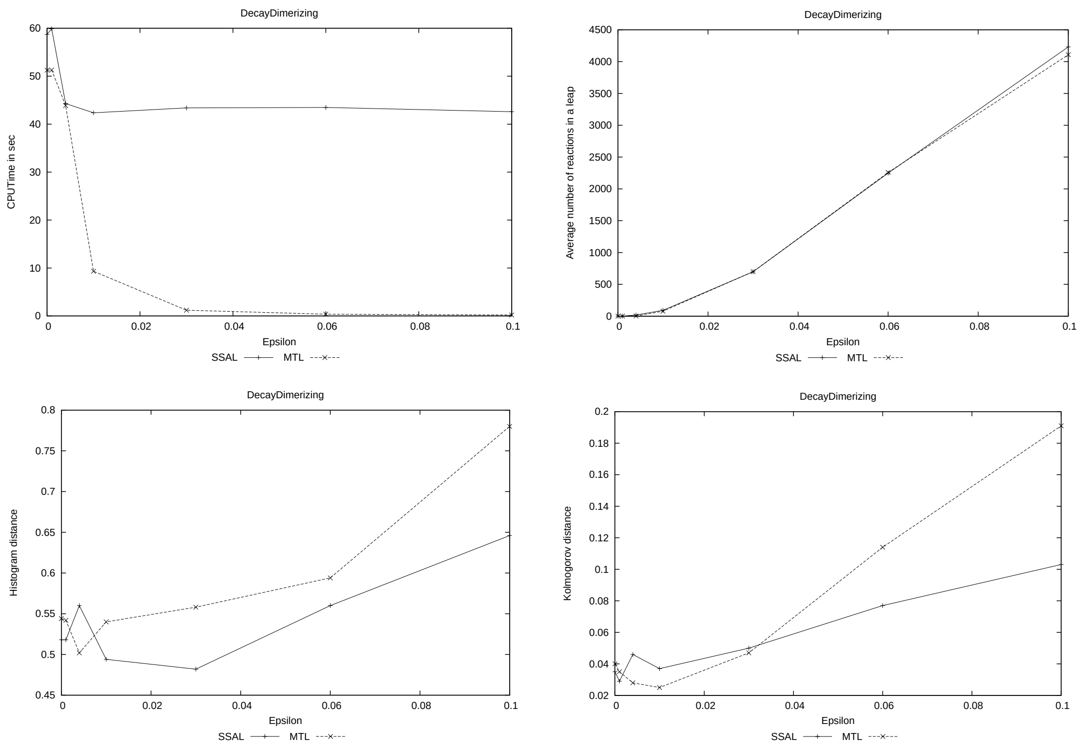


Figure 10.1: Results comparison between SSAL and MTL for 1000 independent simulations for the Decaying-Dimerizing reactions. Accuracy have been taken for the values $X_0(10.0)$.

$c_4 = 0.04$ and the initial state $\mathbf{X}(t_0) = (\mathbf{x}_1 = 4150, \mathbf{x}_2 = 39565, \mathbf{x}_3 = 3445)$. We set also stop time $TIME = 10$, $n_c = 10$, $q = 100$ and $p = 10$. To run 1000 independent simulations, DM required 43.625 seconds. Fig. 10.1 and Table 10.1 summarize the results obtained for SSAL and MTL. Fig. 10.1 shows that m' is greater than m'' for small values of ϵ. In particular for $\epsilon = 0.004$. As we can see, SSAL fired $m' = 18.53$ reactions, whereas MTL only $m'' = 3.65$. It can be also noted from Table 10.1 that, for small ϵ values, MTL computes many shifts. Being $\frac{\log(M)n}{M} = \frac{279036}{2} = 139138$ and $n''' \approx n' * q$, we have that $n''' > \frac{\log(M)n}{M}$ for any value $\epsilon \leq 0.001$. Although theoretically for $n''' > \frac{\log(M)n}{M}$ SSAL ought to perform better than MTL, for this specific biological model, the CPU time of MTL results always smaller than the CPU time of SSAL ($CPUTimeSSAL > CPUTimeMTL$), as we can see in Fig. 10.1. This happens because the model involves only four reactions, so it takes the same time to SSAL to compute and sum $M = 4$ propensity functions and to verify the leap condition. For the same reason, the CPU time of SSAL and DM almost coincide for $\epsilon > 0.004$.
Apart for $\epsilon = 0.1$, the histogram and Kolmogorov distances are very close to the histogram and Kolmogorov self distances provided.

ϵ	$n''' + n''$	n'	$\tau' < \tau''$	$\tau'' < \tau'$
0.0	279036.8	2790.85	0	0
0.001	279000.0	2790.5	0	0
0.004	76264.4	613.69	14943.8	0
0.01	3564.1	0.21	2562.8	0
0.03	399.2	0.027	399.1	0
0.06	123.1	0.011	123.83	0
0.1	68.39	0.003	68.37	0

Table 10.1: MTL statistics for the Decaying-Dimerizing reactions for 1000 independent run.

10.8 Map Kinase Cascade Model

Recently Chatterjee et. al [50] applied their Binomial τ-leaping method to the signaling pathway of epidermal growth factor (EGF) receptor (EGFR) activated mitogen activated protein (MAP) kinase cascade. EGFRs belong to the receptor tyrosine kinase (RTK) family of receptors and play an important role in many physiological processes among which cell proliferation. This biological model involves 106 species and 296 reactions. The reaction set, initial amount and kinetic coefficients have been taken from the web site `http://www.dion.che.udel.edu/multiscale/software.html`. Addditionally, we set stop time $TIME = 0.1$, $n_c = 10$, $q = 100$ and $p = 10$. To run 1000 independent simulations, DM required 209.53 seconds. Fig. 10.2 and 10.2 summarize the results for SSAL and MTL. Fig. 10.2 shows that m' is greater than m'' for $\epsilon \leq 0.01$. Then being $\frac{\log(M)n}{M} = \frac{8.2*60237}{296} = 1668.7$ and $n''' \approx n' * q$, we have $n''' > \frac{\log(M)n}{M}$ for $\epsilon \leq 0.01$. Although theoretically when $n''' < \frac{\log(M)n}{M}$ MTL ought to perform better than SSAL, Fig. 10.2 shows that SSAL performs better than MTL in that range as well. This happens because the extra costs of MTL requires substantial computational extra time for $M = 296$ and $N = 106$. In Table 10.2 this has a peak in correspondence of $\epsilon = 0.001$. SSAL is almost one order of magnitude faster. Here, MTL executes $\tau' < \tau'' = 3515.7$ times the τ-leaping branch, but the number of reaction fired in each of these leap is very small. This means that MTL computes τ' for $M = 296$, but unfortunately often the leap fires only one reaction. This case is a realistic example of the impact of the extra operations of MTL in the simulation time. Again the accuracy is very close to the self distances both for SSAL and MTL.

10.9 LacZ/LacY Model

This model was first introduced by Kierzek et al. in [99] but we consider the model reviewed in [131]. It consists of 23 species (N=23) and 22 reactions (M=22), details and kinetic parameters can be found in those references. In this model the cell vol-

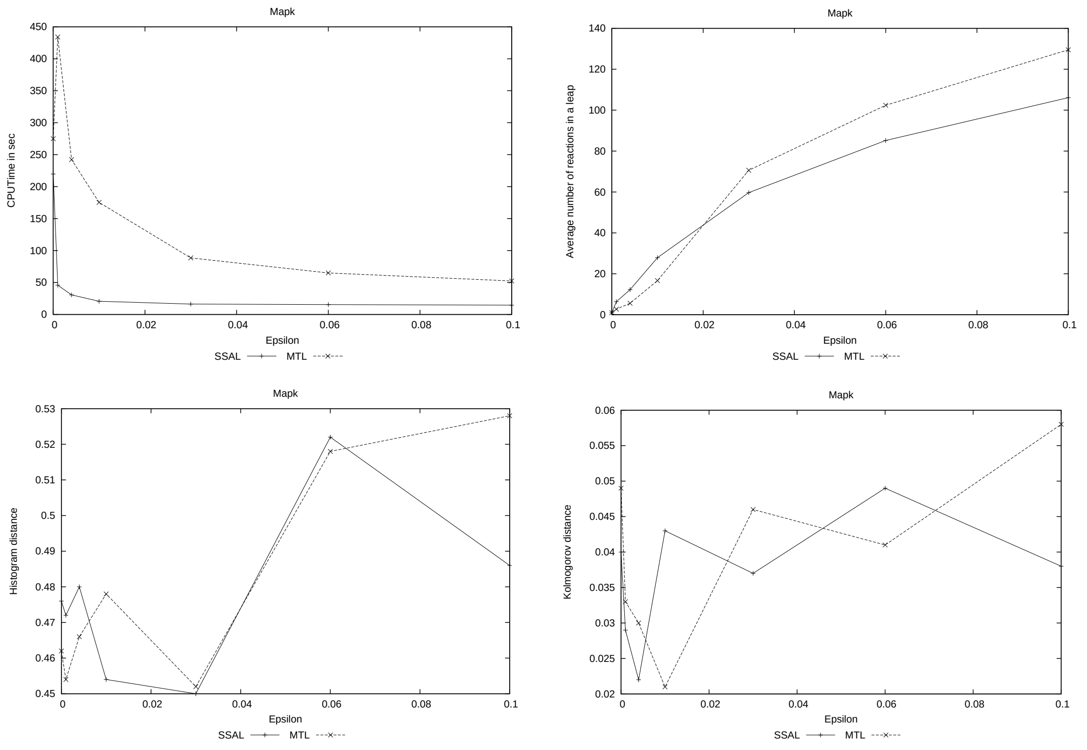


Figure 10.2: Results comparison between SSAL and MTL for 1000 independent simulations for the Map Kinase Cascade reactions. Accuracy have been taken for the values $X_2(0.1)$.

ϵ	$n''' + n''$	n'	$\tau' < \tau''$	$\tau'' < \tau'$
0.0	60237	608.9	0	0
0.001	22266.1	189.87	3515.7	0.82
0.004	10742.3	88.58	2018.6	1.94
0.01	3725.3	22.24	1435.6	136.86
0.03	853.3	0.65	625.3	223.5
0.06	588.4	0.63	362.3	221.8
0.1	465.2	0.65	269.12	224.49

Table 10.2: MTL statistics for the Map Kinase Cascade reactions for 1000 independent run.

ume is assumed to grow in time according to the formula $V(t) = V_0(1+t/T_{gen})$. The initial cell volume is $V_0 = 10^{-15}$ liter and $T_{gen} = 2100$ seconds. Than the population of the two species RNAP and Ribosome are randomly determined from the two Normal random variables $N(35 * (1 + t/2100), 3.5)$ and $N(350 * (1 + t/2100), 35)$. The mean values of these variables grow together with the volume of the cell so that the concentrations of these molecules remain constant [163]. We simulate this system in the time interval $[300, 330]$ for two reasons. The first is that consider also the [0,300] takes too much time. In fact, 1000 independent simulations of DM required 4840 seconds and $n = 6.409324e^6$. The second is because in literature the time interval [300,330] is well studied [21, 131, 163]. For these reasons, we simulated DM in the interval [0,300], we have taken the state of the simulation at time $t = 300$ and we used it as initial state for the SSAL and MTL simulations. MTL used the parameters: $n_c = 10$, $q = 100$ and $p = 10$. For this model 1000 independent simulations of DM in the interval [300,330] required 2053.1 seconds. Fig. 10.3 and Table 10.3 summarize the results for SSAL and MTL. Fig. 10.3 shows that the number of reaction fired m' is greater than m'' for each value ϵ. In particular, for $\epsilon = 0.03$ the number of reactions fired in a leap for SSAL averages at $m' = 10.75$, instead, for MTL it is $m'' = 1.056$. For the LacZ/LacY model, we have $\frac{\log(M)n}{M} = \frac{4.45*2877222}{22} = 581983.5$ and $n''' \approx n' * q$. Whereas it results that $n''' > \frac{\log(M)n}{M}$ for $\epsilon \leq 0.03$. Again CPU times show that SSAL performs better than MTL for any value ϵ in the table. In particular, SSAL is four times faster for $\epsilon = 0.03$. Accuracy of the results in Fig. 10.3 are very similar for SSAL and MTL. Table 10.3 shows that for small ϵ MTL shifts to DM, while in the other cases the simulation turns into the branches $\tau' < \tau''$ and $\tau'' < \tau'$. We omit the negative population branch in all the tables because no negative populations occurred for the parameters chosen for the models.

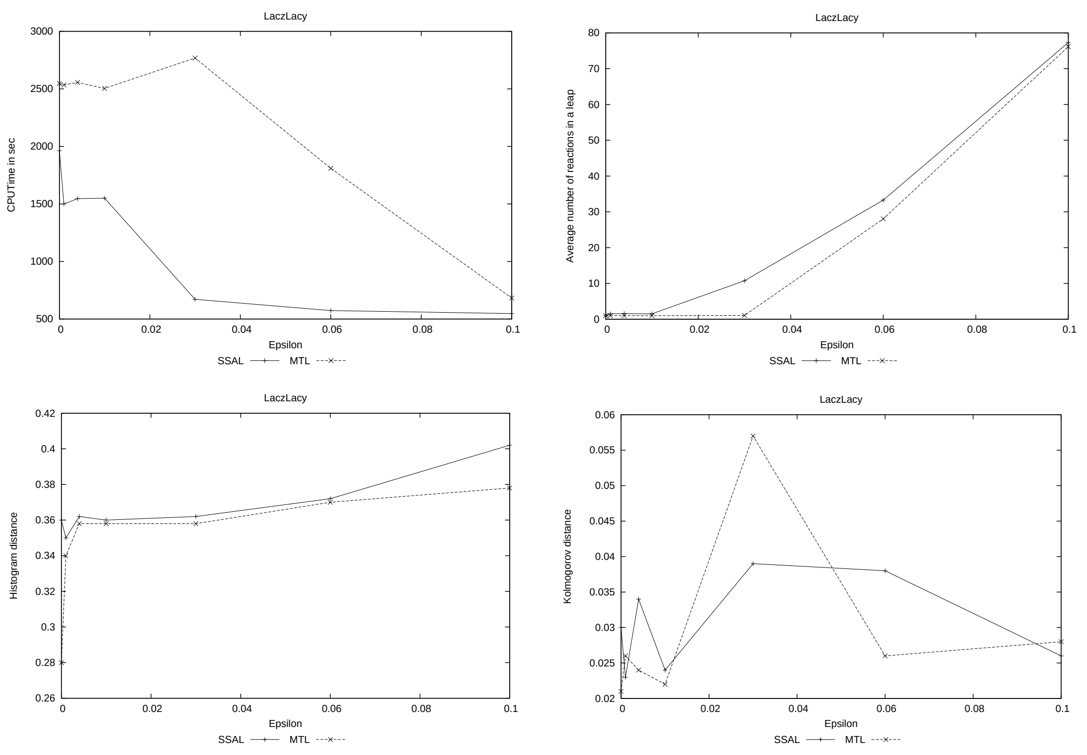


Figure 10.3: Results comparison between SSAL and MTL for 1000 independent simulations for the LacZ/LacY reactions. Accuracy have been taken for the values $X_{trrbslacy}(330)$.

ϵ	$n''' + n''$	n'	$\tau' < \tau''$	$\tau'' < \tau'$
0.0	2877222	28772.6	0	0
0.001	2871591	28716.3	0	0
0.004	2880443	28736.8	0	0
0.01	2723601	28738.6	0	0
0.03	2724048	27064	14621.7	125.4
0.06	102529.3	10.8	99298.2	2405.4
0.1	37771	0.097	35386.5	2409.7

Table 10.3: MTL statistics for the LacZ/LacY reactions for 1000 independent run.

10.10 Performance Comparison

Now, to conclude we provide the speed-up of SSAL against DM and MTL, respectively. The speed-up is taken by dividing the CPU Time of a method for the CPU Time of SSAL. Fig. 10.4 shows that, apart for some ϵ in the Decaying-Dimerizing model, SSAL performs better than DM. While Fig. 10.5 confirms that SSAL performs better than MTL in the non trivial cases.

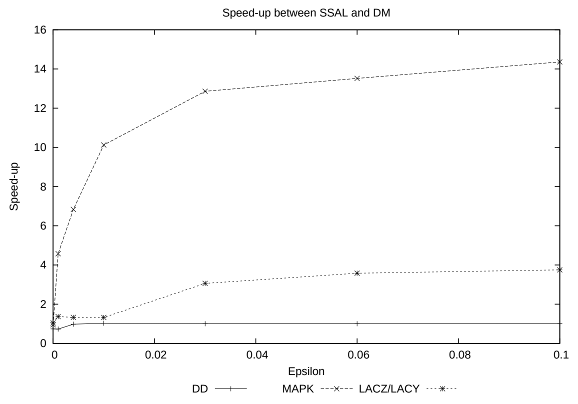


Figure 10.4: Speed-up between SSAL and DM.

10.11 Conclusions

We presented SSAL, a new method which lays in the middle between the direct method (DM) and a τ-leaping. The SSALeaping method *adaptively* builds leap and stepwise updates the system state. Differently from MTL, SSAL neither shifts from τ-leaping to DM nor pre-selects the time leap τ'. Additionally whereas MTL prevents negative populations taking apart critical and non critical reactions, SSAL generates sequentially the reactions to fire verifying the leap condition after each generation. We proved that a reaction overdraws one of its reactants if and only if

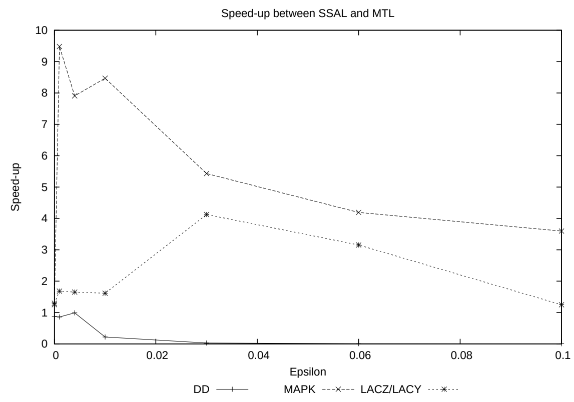


Figure 10.5: Speed-up between SSAL and MTL.

the leap condition is violated. Therefore, this makes it impossible for the population to become negatives, because SSAL stops the leap generation in advance.
In order to compare SSAL with existing methods, we focused on the Modified τ-leaping (MTL) and the direct method (DM). For them we provided the time complexity and a detailed asymptotic analysis. These allowed to abstract from many implementation details and model specifications, and it highlights both the bottleneck operations and the specific features of the model that make these methods inefficient. We showed that in the worst case, the complexity of MTL and SSAL reduce themselves to that of DM. Instead, SSAL performs better than MTL if the number of reactions fired sequentially by MTL exceeds the bound $\frac{\log(M)n}{M}$.

We also integrated the analysis with some numerical experiments to test how the above methods work in practice. We run our implementations of SSAL, DM and MTL upon the Decaying-Dimerizing, the Map Kinase Cascade and the LacZ/LacY models. Results substantially agrees with the theoretical analysis provided. They showed that for MAPK and LacZ/LacY and for $\epsilon > 0$ our SSAL implementation performs better than the two implementations of MTL and DM. Additionally, they highlighted that $n''' > \frac{\log(M)n}{M}$ is the main range emerged for two of the three realistic biological models considered. In this range SSAL performs better than MTL.

Concluding results confirmed that SSAL is very promising to simulate realistic biological models. We noted that between MTL and SSAL exists a very clear complementarity because SSAL seems to perform better than MTL when the the number of reactions fired in a leap is small compared to the model size M, otherwise MTL performs better than SSAL in the other cases. For the future we will investigate a solution that expect to gain efficiency from this complementarity.

Chapter 11

Efficient Adaptive Tau-leaping Method

The main differences between SSAL and MTL can be summarized as follows. On one hand, SSAL avoids any leap pre-computation, and it builds leap incrementally by quickly verifying the leap condition for the selected reaction. This means that SSAL pays a cost proportional to the number of reactions effectively fired for each leap. On the other hand, MTL computes some extra costs at the beginning of the leap and it samples the number of occurrences of each reaction. The computation of such extra information regards operations on the total number of reactions in the system independently on the number of reactions that will fire. This means that for each leap MTL pays a cost proportional to the number of reactions in the system. Comparing the characteristics of SSAL and MTL it is arguable a natural complementarity between them that is related to the number of reactions that will fire and the number of reactions of the system. In this chapter we propose a new adaptive method, called *Adaptive Modified SSALeaping* (AMS), that exploits the complementarity between SSAL and MTL. This method adaptively switches from one method to the other and viceversa during the simulation. In this way, we expect to be able augment the number of systems eligible to be simulated efficiently. But to fully exploit the idea AMS is based on we require to solve two issues: how identify the threshold μ that fixes the point of switching, and how to estimate the number of reactions k' that will fire in a future leap that is necessary to decide the switch for a specific leap. In this chapter we introduce AMS and its main features. We start by proving complementarity in a rigorous way, we introduce two proposed solutions for the issues mentioned above and we present AMS and a bunch of numerical experiments.

11.1 Adaptive Modified SSALeaping (AMS)

To show complementarity we use a simple asymptotical analysis. We start from the complexity T_{SSAL} in Eq. 10.17 and T_{MTL} in Eq. 10.9. We restrict these complexities in such a way that they refer to a single leap. To further simplify the analysis we consider only the case in which the leap is long enough that MTL has not need to switch to DM. This means that we further restricted the complexity T_{MTL} by eliminating the term $(C_{TSTEP} + C_{DM})n'$. We summarize the resulting costs in Eq. 11.1 for SSAL and Eq. 11.2 for MTL, respectively.

$$T_{SSAL} = (C_{a_j} + C_{a_0} + C_{copy}) + (C_{2r} + C_\tau + C_j + C_{update} + C_{Leap})k' \tag{11.1}$$

$$T_{MTL} = (C_{TSTEP} + C_{TLEAP}) \tag{11.2}$$

The costs C_{TSTEP} and C_{TLEAP} in Eq. 11.2 are the same costs defined in Eq. 10.8 and in Eq. 10.7.

Now as the complementarity depends by the number of reactions fired in a leap we want to show the order of magnitude of k' that makes the complexity T_{SSAL} better, equal or worst than the complexity T_{MTL}.

To simplify the comparison we eliminate the costs C_{2r}, C_τ, C_{update}, C_{Leap} from T_{SSAL} because they are asymptotically constants.

$$\begin{aligned} T_{SSAL} &< T_{MTL} \\ (C_{a_j} + C_{a_0} + C_{copy}) + (C_{2r} + C_\tau + C_j + C_{update} + C_{Leap})k' &< T_{MTL} \\ C_{copy} + (C_j)k' &< T_{MTL} - C_{a_j} - C_{a_0} \end{aligned} \tag{11.3}$$

Now we substitute the asymptotic complexities associated to the corresponding costs and we isolate the order for k'.

$$\begin{aligned} C_j k' &< C_L + C_{\tau'} + \max\{C_{\tau'<\tau''}, C_{\tau'\geq\tau''}\} + C_{neg} + C_{update} - C_{copy} \\ \Theta(\log(M)k') &< \Theta(M) \\ k' &< \Theta(\frac{M}{\log(M)}) \end{aligned} \tag{11.4}$$

It results that T_{SSAL} is asymptotically better than T_{MTL} for $k' < \Theta(\frac{M}{\log(M)})$. T_{SSAL} and T_{MTL} results comparable for $k' = \Theta(\frac{M}{\log(M)})$ and T_{SSAL} performs asymptotically worst than T_{MTL} for $k' > \Theta(\frac{M}{\log(M)})$. This gives important indications on the order of magnitude of the threshold μ that separates asymptotically the cases in which SSAL or MTL is faster than the other. They also prove the complementarity existing between SSAL and MTL. In accordance to the asymptotic analysis we have $\mu = \Theta(\frac{M}{\log(M)})$.

The following sections take into account how determine threshold value and how to estimate the number of reactions that will fire in a leap.

11.1.1 Determining the threshold

For what concern the threshold the preceding asymptotic analysis abstracts from many implementation details and model specifications, highlighting costly operations and most relations among those operations and simulation parameters. However, some of the costs that can asymptotically be neglected, in practice can be a burden. In particular, the smaller the number of reactions in the system, the more burden these costs have. Only to give some simple example of some of these costs given a very small system consider the processing of the cumulative τ; or the generation of the uniform random numbers for SSAL; or the generation of the poisson random numbers for MTL. Give a precise and general quantification of the weigh introduced by these operations can be a very hard task. This is because any quantification is impossible without information about hardware specifications and software implementations. For this reason we assign the value $c\frac{M}{\log(M)}$ to the threshold μ, letting the quantification of a specific value for the constant c experimentally.

11.1.2 Estimating the occurrences in a leap

For what concern the estimation of the number of reactions that will fire in a leap, in our asymptotic analysis we discussed the values of k' and the threshold μ for which we have better or worst complexities using SSAL and MTL. However, in practice k' is unknown at the beginning of the leap.

MTL has a simple way to compute k' explicitly. This is given by the formula in Eq. 10.6. According to this formula we just need to select the closest integer to the sample generated by the distribution $\mathcal{P}(\tau' a_0)$. This requires to know the values a_0 and τ'. So if AMS is executing MTL this is not a problem because the computation of τ' is part of the algorithm. However, since SSAL avoids the computation of τ' at the begin of the leap, using the formula to estimate k' is impossible when AMS executes SSAL. Furthermore, we want to avoid the explicit computation of τ' using some explicit τ-selection procedure. So we need to find something different for SSAL.

To this end we propose to estimate k' as follows. Suppose we are at the i-th leap of the simulation, our solution guesses k' by approximating it with the average number of reactions effectively fired until that moment. Mathematically, let n_i the number of reactions effectively fired until the $i-1$-th leap, we define k_{mean} as the following average value.

$$k_{mean} = \frac{n_i}{i-1} \tag{11.5}$$

The k_{mean} formula in Eq. 11.5 is clearly an approximation of k'. Intuitively, given model the simulation can be faster by using SSAL or MTL. In this cases k_{mean} will be much smaller or larger than the threshold and this will be well recognized by AMS. Instead, for those cases in which k_{mean} results very close to the threshold both methods are suited because around the threshold they have the same computational costs and consequently execution times. However, we need to clarify that at the moment AMS is not able to select the fastest method at each leap but only the fastest according to the trend. This is a very interesting aspect of our method and for now it is under investigation and we expect to propose some alternative in the future. For now we focus on the preceding proposal based on k_{mean}. The introduction of k_{mean} involves rewrite the following new switch condition.

$$k_{mean} > \mu \tag{11.6}$$

The condition must be interpreted as follows. If k_{mean} is smaller than the threshold μ execute SSAL for the leap under consideration, otherwise execute MTL.

11.1.3 Algorithmic Optimizations

In our investigation we also identified two important optimizations that we introduce as follows. The first regards the typical switch to DM of MTL. This operation has been introduced to efficiently deal with small leaps. In particular, leaps for which $\tau' < 10/a_0$, where τ' is the time interval obtained by the τ-selection procedure and $1/a_0$ is the mean waiting time for the next reaction to fire. In other words, when the mean number of reactions expected to fire in τ' are less or equal to 10. Now as we proved that SSAL is more efficient than DM, we substitute DM with SSAL in the switch operation of MTL. The new switch to SSAL requires to change the switch condition inequality. We substituted the condition in Eq. 10.6 into the new condition in Eq. 11.7.

$$\mathcal{P}(\tau' a_0(\mathbf{x})) < \mu \tag{11.7}$$

Note that we do not only consider the mean value $a_0\tau$ as proposed in the original version of MTL. We estimate the number of reactions that will fire by sampling from a Poisson distribution to assure more accurate estimation of k' also when the mean $\tau' a_0(\mathbf{x})$ is small.

The second optimization regards the computation of the cumulative τ' in SSAL. Until now SSAL incrementally computes the value of τ' by summing sample times generated from an exponential distribution with parameter a_0 each time a new reaction occurs. For very small leap this method is efficient, but if the number of reactions that fire in a leap increases, more efficient methods exist. These methods exploit the well known relation between Exponential and Erlang distributions. Erlang distribution $Erl(l, \theta)$ represents the sum of l independent exponentially distributed random variables, each of which has mean θ. This means that giving the number of reactions fired by SSAL in a leap k', we compute the time interval τ' for a leap by drawing a sample taken from an Erlang distribution with shape parameter k' and scale parameter a_0^{-1}:

$$\tau' \approx Erl(k', a_0^{-1}) \tag{11.8}$$

The use of the Erlang distribution to determine τ' has been used by a method called R-leaping method [21]. The difference between the use of Erlang in R-leaping and in

SSAL is that R-leaping selects the shape parameter θ at the beginning of the leap, whereas SSAL computes this value at the end of the leap, in other words ones that SSAL knows the exact number of reactions fired.

To efficiently generate samples from the Erlang when $l < 1000$ exist very fast methods, for example Cheng/Feast Algorithm [183]. Then when $k' > 1000$ Erlang distribution is well approximated by the Gaussian distribution, and sampling from it becomes computationally even more efficient generators have been proposed. Summing up, using Erlang (approximated and not) we further improved the performance of SSAL and consequently of our adaptive method.

11.1.4 The Algorithm AMS

Identified a formula for the threshold, the estimation of the number of reactions fired and some optimizations we can introduce AMS in detail. The adaptive modified SSALeaping is a sort of meta-algorithm that decides the procedure to use for each leap. Intuitively, AMS can be divided into three steps. The first evaluates the algorithm to use. The switch between one method and the other entirely depends by the values k_{mean} and μ. If $k_{mean} \leq \mu$ then SSAL is expected to be faster. If $k_{mean} > \mu$ then MTL is expected to be faster. In the second step AMS applies the chosen method by executing a procedure implementing the simulation method corresponding. In the third AMS updates k_{mean} according to the effective number of reactions fired and the formula in Eq. 11.5.

So giving M reactions and kinetic constants, N species, one initial state $X(t_0)$ and a stop time TIME, two user-defined parameter n_c and ϵ, AMS performs the elementary steps in Alg. 8. The procedure Alg. 8 calls the procedures Alg. 9 and Alg. 10, where Alg. 9 implements SSAL with the new computation of τ according to the formula in Eq. 11.8. Whereas Alg. 10 implements MTL adapted with the new switch condition introduced in Eq. 11.7. Both procedures update the value of k_{mean}.

Taking into account our optimizations and assuming N of order of magnitude of M, we show the new time complexities for SSAL and MTL in Eq. 11.9 and Eq. 11.10, respectively. The costs we identified are the following.

1. C_{a_j}, C_{a_0}, C_{copy}, C_j, C_{Leap}, C_{update}, the same of SSAL
2. C_τ is the cost to sample τ
3. C_r is the cost to generate one uniformly distributed random number in [0,1)
4. C_L, $C_{\tau'}$, $C_{\tau''}$, $C_{\tau'<\tau''}$, $C_{\tau'\geq\tau''}$, C_{neg} are the same of MTL
5. $C_{k_{mean}}^{SSAL}$, $C_{k_{mean}}^{MTL}$ are the costs to compute k_{mean} in SSAL and in MTL, respectively.

Algorithm 8 Adaptive Modified SSALeaping

```
$\mu \leftarrow c\frac{M}{\log(M)}$
$n_i \leftarrow 0$;
$k_i \leftarrow 0$
$k_{mean} \leftarrow 0$
**while** $t < TIME$ **do**
    Compute $a_1(\mathbf{x})$
    **for** j=2 to M **do**
        Compute $a_j(\mathbf{x})$
        $A[j] \leftarrow A[j-1] + a_j(\mathbf{x})$
    **end for**
    $a_0 \leftarrow A[M-1]$
    **if** $k_{mean} \leq \mu$ **then**
        Execute SSAL through the procedure Alg. 9
    **else**
        Execute MTL through the procedure Alg. 10
    **end if**
**end while**
```

Algorithm 9 SSALeaping Leap Generation Procedure

```
Store a copy of $\mathbf{x}(t)$
$OK \leftarrow true$
$\tau' \leftarrow 0$
**while** $OK = true$ **do**
    Generate $r_1$ in $U(0,1)$, and find $j$ according to Eq. 5.7 through binary search.
    $n_i \leftarrow n_i + 1$;
    $\mathbf{x}' \leftarrow \mathbf{x}' + \nu_j$;
    **for all** $i \in ((Reactants(R_j) \cup Products(R_j)) \cap I_{rs})$ **do**
        **if** $\mid \mathbf{x}'_i - \mathbf{x}_i \mid > \epsilon_i \mathbf{x}_i$ **then**
            $OK \leftarrow false$
        **end if**
    **end for**
**end while**
$k_i \leftarrow k_i + 1$;
Generate a sample $\tau$ according to the distribution in Eq. 11.8.
$t \leftarrow t + \tau$
compute $k_{mean} = \frac{n_i}{k_i}$
```

Algorithm 10 Modified τ-leaping Leap Generation procedure

```
for j=1 to M do
    Compute $L_j$ according to Eq. 10.4
    if $L_j < n_c$ then
        $J_c \leftarrow J_c \cup j$
    else
        $J_{nc} \leftarrow J_{nc} \cup j$
    end if
end for
if $J_{nc} \neq \{\}$ then
    Compute $\tau'$ according to Eq. 10.5
else
    $\tau' \leftarrow 0$
end if
repeat
    if $\mu > \mathcal{P}(\tau' a_0(\mathbf{x}))$ then
        $SWITCH \leftarrow SSAL$
        run SSAL through the procedure Alg. 9
    else
        if $J_c \neq \{\}$ then
            Compute $\tau'' = \frac{1}{a0_c(\mathbf{x})} \ln\left(\frac{1}{r_1}\right)$
        else
            $\tau'' = 0$
        end if
        Execute CompareTimes in Alg. 6
        $temp \leftarrow t$
        $t \leftarrow t + \tau$
        $\mathbf{x}_i \leftarrow \mathbf{x}_i + \sum_{j=1}^{M} \nu_{ij} k_j$
        for i from 1 to N do
            if $\mathbf{x}_i < 0$ then
                $\mathbf{x}_i \leftarrow \mathbf{x}_i - \sum_{j=1}^{M} \nu_{ij} k_j$
                $t \leftarrow t - \tau;$
                $\tau' = \tau'/2$
            end if
        end for
    end if
until $t = temp$
$n_i \leftarrow n_i + \sum_{j=1}^{M} k_j;$
$k_i \leftarrow k_i + 1;$
compute $k_{mean} = \frac{n_i}{k_i}$
```

The cost $C^{SSAL}_{k_{mean}}$ is of order of magnitude $O(1)$ because k_{mean} only requires one increment for each step and one division. Instead, the cost $C^{MTL}_{k_{mean}}$ results $\Theta(M)$ because it requires to sum M k_j's. Summing up, Eq. 11.9 and Eq. 11.10 summarize the complexities for SSAL and MTL, respectively.

$$\begin{aligned} T'_{SSAL} &= (C_{a_j} + C_{a_0} + C_{copy} + C_r + C_\tau)k + (C_r + C_j + C_{update} + C_{Leap} + C^{SSAL}_{k_{mean}})n \\ &= \Theta(Mk + \log_2(M)n). \end{aligned} \tag{11.9}$$

The main differences between the original version T_{SSAL} and T'_{SSAL} are the following. First, the cost C_τ is now processed at the end of the leap instead that for each step. Second, $C^{SSAL}_{k_{mean}}$ belongs now to the costs paid for each step of the simulation. The cost C_{2r} has been substituted by C_r. However, in spite of the new optimizations the complexity of SSAL remains the same.

$$\begin{aligned} T'_{MTL} &= (C_{TSTEP} + T'_{SSAL})n' + (C_{TSTEP} + C_{TLEAP} + C^{MTL}_{k_{mean}})n'' \\ &= \Theta(\log_2(M)k'n' + Mn' + Mn'') \\ &= \Theta(\log_2(M)n''' + Mn' + Mn''). \end{aligned} \tag{11.10}$$

In this case, the main difference with the complexity in Eq. 10.9 is that the switch to DM has been substituted with the switch to SSAL. Consequently, instead to have q steps we have the number of steps k' computed by SSAL. Asymptotically the complexity T'_{MTL} is better than T_{MTL} when $n''' > n'$. While T'_{MTL} and T_{MTL} have the same complexity when $n''' \leq n'$. In this last case few reactions have been fired by SSAL in average per leap.

Finally, starting from the two time complexities defined before for SSAL and MTL, the asymptotic time complexity for AMS is the sum of T'_{SSAL} and T'_{MTL}.

$$T_{AMS} = T'_{SSAL} + T'_{MTL} \tag{11.11}$$

If any shift to SSAL occurs during the simulation (i.e. k=n=0) the complexity $T_{AMS} = T'_{MTL}$. Whereas if any shift to MTL occurs during the simulation (i.e. $n' = n'' = n''' = 0$) the complexity $T_{AMS} = T'_{SSAL}$.

11.2 Numerical Experiments

Now we intend to give practical evidence of the efficiency and accuracy of our method by providing numerical experiments. We selected three known realistic biological models that are: the Decaying-Dimerizing, the epidermal growth factor (EGF) receptor (EGFR) activated mitogen activated protein (MAP) kinase cascade and the LacZ/LacY. For each biological model we run our C language implementation of AMS, SSAL and MTL ranging ϵ in the set (0.003, 0.006, 0.009, 0.012, $\cdots$, 0.06) and the threshold μ in the set ($4M/\log(M)$, $8M/\log(M)$, $16M/\log(M)$). For each combination (μ, ϵ) we run a number of independent simulations and we collect information about CPU Time ($CPUTime$), average number of reactions fired in a leap (k'), number of leap computed with SSAL (k) and number of leap computed with MTL (n''). In this way, we expect to identify the constant value for c in $\mu = c\frac{M}{\log(M)}$ for which AMS obtains the most efficient simulations. We also estimated accuracy of results of AMS, SSAL and MTL by computing first the histogram and then Kolmogorov distances [47] for ensembles of 1000 independent simulations. For each experiment, we provided the relative Histogram and Kolmogorov distances between the results of DM and AMS, SSAL and MTL. Additionally, we computed the so called *self distance* in order to have an absolute comparison value for the accuracy [47]. We interpreted the results as follows. The closer to the self distance a distance value is, the more accurate the method who has generated that samples will be. For ensembles of 1000 simulations the mean and variance for the histogram self distance are 0.079 and 0.49, respectively. Whereas for the Kolmogorov self distance we have 0.0389 and 0.00136, respectively. Then all experiments run in a WINDOWS XP personal computer with a 3.0 GHz CPU and 1 Gbyte memory.

11.2.1 Decaying-Dimerizing Model

This first test model is the same seen in Chap. 10 with three species S_1, S_2 and S_3 (N=3) and four reactions (M=4). We simulate the model using the following stochastic coefficients: $c_1 = 1.0$, $c_2 = 0.002$, $c_3 = 0.5$ and $c_4 = 0.04$ and the initial state $\mathbf{X}(t_0) = (\mathbf{x}_1 = 4150, \mathbf{x}_2 = 39565, \mathbf{x}_3 = 3445)$. We set also stop time $TIME = 10$. To run 1000 independent simulations, DM required 43.06 seconds for $2.789923e + 005$ steps. For this model the average number of reactions fired in a leap spans from M for $\epsilon = 0.003$ to more than 500 times M for $\epsilon = 0.06$, as shown in Fig. 11.1. This makes the system well suited for MTL because for most ϵ it results $k' \gg \mu$. For what concerns the CPUTime in Fig. 11.2 all values of μ obtained comparable results. Then if we consider the case $\mu = 16M/\log(M)$, results in Fig. 11.3 show that only for $\epsilon = 0.003$ AMS executes the majority of leaps with SSAL. In the other cases MTL is always the method selected.

11.2.2 LacZ/LacY Model

The second model is the LacZ/LacY model seen in Chap. 10. It consists of 23 species (N=23) and 22 reactions (M=22). For this model, we set up two different experiments for two different time intervals. The first considers the well studied time interval $[300, 330]$ [21, 131, 163]. We simulated DM in the interval [0,300], we have taken the state of the simulation at time $t = 300$, and we used it as initial state for the interval $[300, 330]$. We simulate the model using the following initial state $\mathbf{X}(300) = $ (1, 39, 0, 0, 16, 0, 0, 449, 0, 0, 46, 40, 991, 438, 1964, 2047, 175097, 53, 14, 10, 33, 23, 2275413). For this first range 1000 independent simulations of DM required 1998.64 seconds for $2.873994e + 006$ steps. In this case the average

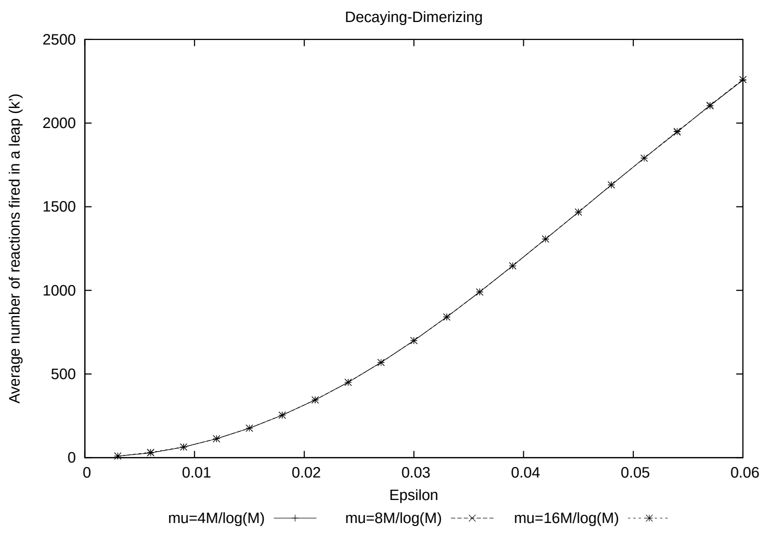


Figure 11.1: Average number of reactions fired in a leap after 1000 independent simulations varying ϵ from 0.003 to 0.06 and μ in the set $(4M/\log(M)$, $8M/\log(M)$, $16M/\log(M))$ for DD model.

number of reactions fired in a leap spans by a small constant to less of two times M, as we can see by Fig. 11.4. This features make this model very suitable for SSAL because for most ϵ $k' < \mu$. The CPUTime in Fig. 11.5 confirms that anticipating the execution of MTL can affect very much execution time. However, the most efficient simulations are for $\mu = 8M/\log(M)$ and $\mu = 16M/\log(M)$. For $\mu = 16M/\log(M)$ SSAL occurs all the time, as we can see in Fig. 11.6.
The second experiment considers the time interval $[600, 700]$. We simulated DM in the interval $[0, 600]$, we took the state of the simulation at time $t = 600$, and we used it as initial state for the AMS simulations. We simulate the model using the following initial state $\mathbf{X}(600) = (1, 38, 0, 0, 20, 0, 3, 457, 0, 1, 63, 51, 1518, 508, 7242, 7191, 188387, 232, 109, 103, 98, 89, 21475758)$. For this range 1000 independent simulations of DM required 25797 seconds for $3.418590e{+}007$ steps, and the number of independent simulation have been reduced from 1000 to 10 because otherwise it would require approximatively more than 1 week. For this case the average number of reactions fired in a leap spans from values smaller than M, to values that exceed 12 times M, as summarized in Fig. 11.7. These range of values is interesting because, for small ϵ, we expect that $k' < \mu$, instead for large ϵ, we expect that $k' > \mu$. This means that for small values of ϵ SSAL results intuitively better than MTL. While for large value of ϵ MTL performs better. This makes a very interesting case to test if AMS is able to choose the best method. Fig. 11.8 shows CPUTime. The best performances happens for $\mu = 16M/\log(M)$. Taking $\mu = 16M/\log(M)$ we note that for $\epsilon < 0.03$ the leap are entirely simulated with SSAL, whereas for $\epsilon > 0.03$ the leap are simulated with MTL, as shown by Fig. 11.9.

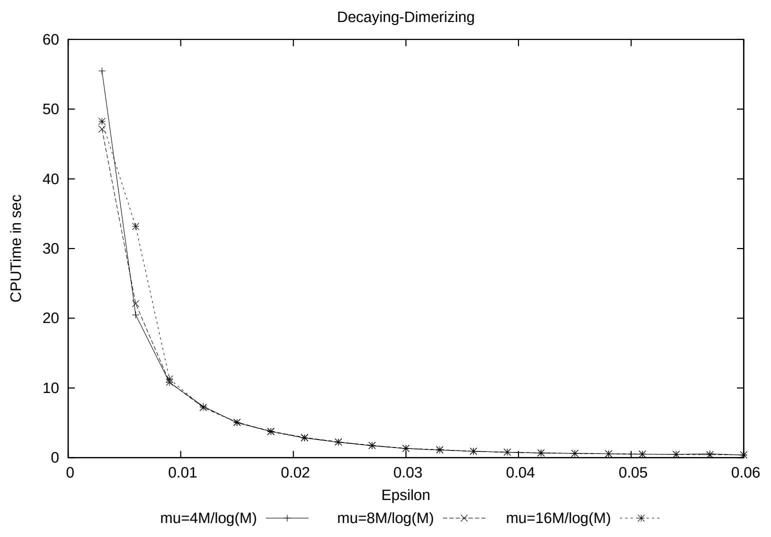


Figure 11.2: CPUTime after 1000 independent simulations varying ϵ from 0.003 to 0.06 and μ in the set $(4M/\log(M),\ 8M/\log(M),\ 16M/\log(M))$ for DD model.

11.2.3 Map Kinase Cascade Model

The last model is the signaling pathway of epidermal growth factor (EGF) receptor (EGFR) activated mitogen activated protein (MAP) kinase cascade, MAPK for short. This biological model involves 106 species and 296 reactions. The reaction set, initial amount and kinetic coefficients are the same seen in Chap. 10. For stop time $TIME = 0.1$, running 1000 independent simulations, DM required 256.5 seconds for $7.543967e+004$ steps. For this model, Fig. 11.10 shows that the average number of reactions fired in a leap spans from very few to less than $M/4$. Then we expect that for all ϵ we obtain $k' < \mu$. If we consider the relatively large number of reactions characterizing this model, the simulation is intuitively well suited for SSAL, and not well suited for MTL. The simulation times in Fig. 11.11 are affected by some fluctuations, but $\mu = 16M/\log(M)$ gives very efficient simulations for any value of ϵ in the range considered. As we expect, in Fig. 11.12 SSAL simulates the total number of leap, whereas MTL never occurs.

11.2.4 Performance and Accuracy Comparison

Now to conclude, we compare the CPUTime between SSAL, MTL and AMS considering the threshold $\mu = 16M/\log(M)$ emerged as the best threshold in the previous experiments. As we can see by Fig. 11.13 AMS often performs better, but most of time as efficiently as the best between SSAL and MTL. As we can see from Fig. 11.14, the relative accuracy of results is perfectly comparable with that obtained with SSAL and MTL. But also those values respect the bounds fixed by the self distances. This confirm that AMS does not add further approximations.

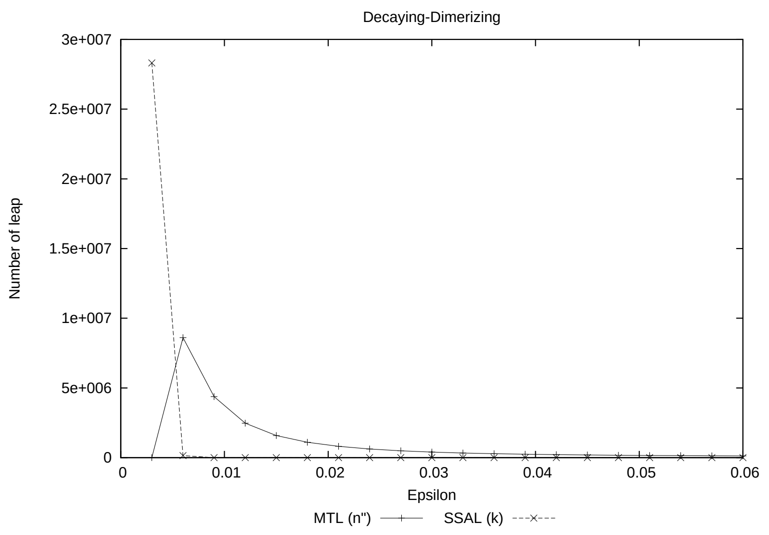


Figure 11.3: Number of leap executed with SSAL and MTL after 1000 independent simulations varying ϵ from 0.003 to 0.06 and $\mu = 16M/\log(M)$ for DD model.

11.3 Conclusions

In this chapter an adaptive method, called *Adaptive Modified SSALeaping* (AMS), has been presented to accelerate the execution of τ-leaping and to augment the number of models eligible to be simulated efficiently. During the simulation, our method switches between SSALeaping (SSAL) and Modified τ-leaping (MTL), according to conditions on the size of the model and the predicted length of the leap. The main issues related to the practical application of the proposed method are: the identification of the threshold μ and the estimation of the number of reactions that will fire at each leap k'. In accordance with asymptotic analysis and numerical experiments we identified the value $\mu = 16M/\log(M)$, as the threshold that produced almost the best efficiency simulations in all the biological models tested. For the second issue, we estimated the number of reactions that will fire at each leap in two ways. If the system is simulating MTL, we adapted the formula used by MTL to switch to DM. If the system is simulating with SSAL, we simply used the average number of reactions fired until the last leap preceding the leap considered. Then AMS avoids negative populations because at each leap, it mutually applies SSAL or MTL, that are well known and effective strategies to solve this issue. The numerical experiments have been conducted upon realistic biological models. The DD and MAPK models are two examples in which SSAL and MTL perform very well in one case but not in both the cases. The LACZ/LACY in the range [600,700] is an example in which for small value of ϵ SSAL performs better than MTL but for larger values the CPUTime required by MTL becomes smaller than that required by SSAL. Results confirmed that AMS performs was able to deal efficiently all the

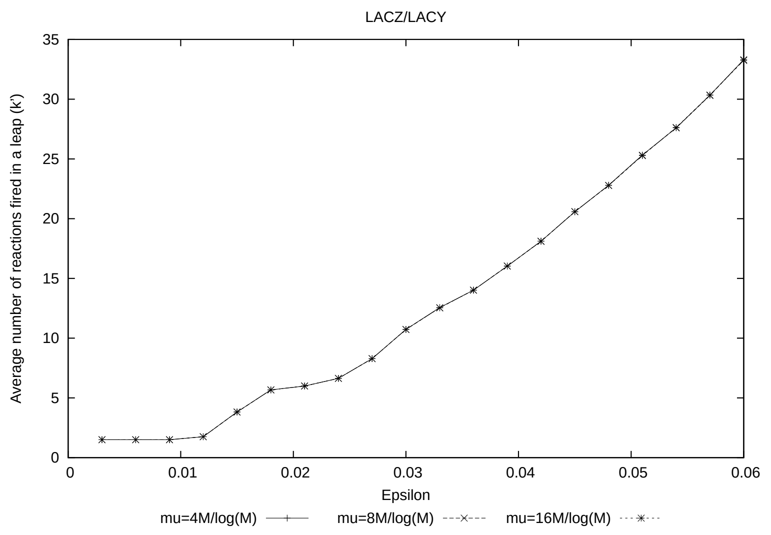


Figure 11.4: Average number of reactions fired in a leap after 100 independent simulations varying ϵ from 0.003 to 0.06 and μ in the set $(4M/\log(M)$, $8M/\log(M)$, $16M/\log(M))$ for LacZ/LacY model in the time interval [300,330].

cases considered. Moreover, maintaining the same accuracy of SSAL and MTL. This prove the adaptivity, efficiency and accuracy of our method.

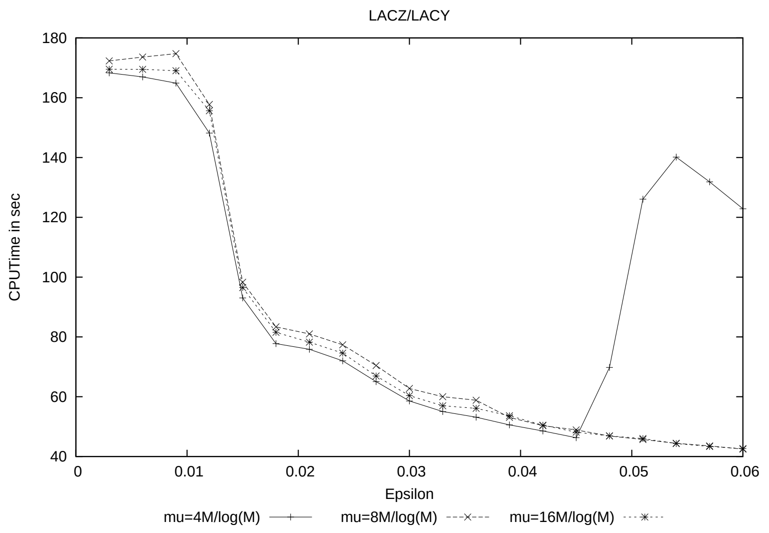


Figure 11.5: CPUTime after 100 independent simulations varying ϵ from 0.003 to 0.06 and μ in the set $(4M/\log(M),\ 8M/\log(M),\ 16M/\log(M))$ for LacZ/LacY model in the time interval [300,330].

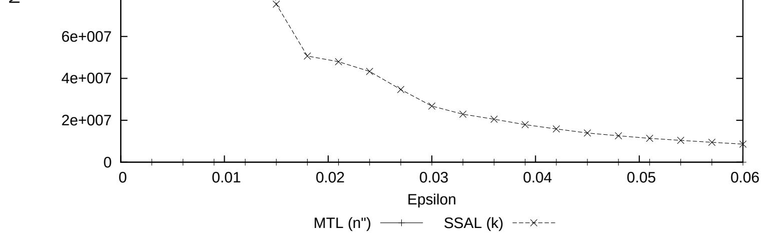


Figure 11.6: Number of leap executed with SSAL and MTL after 100 independent simulations varying ϵ from 0.003 to 0.06 and $\mu = 16M/\log(M)$ for LacZ/LacY model in the time interval [300,330].

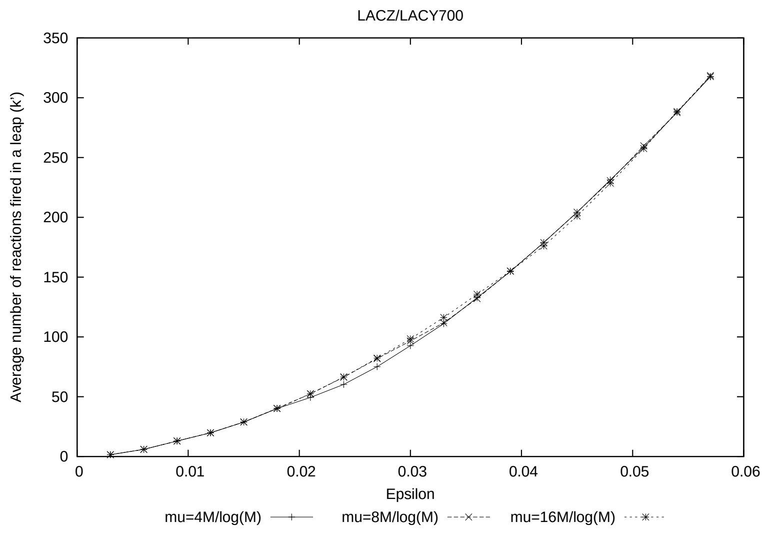


Figure 11.7: Average number of reactions fired in a leap after 10 independent simulations varying ϵ from 0.003 to 0.06 and μ in the set ($4M/\log(M)$, $8M/\log(M)$, $16M/\log(M)$) for LacZ/LacY model in the time interval [600,700].

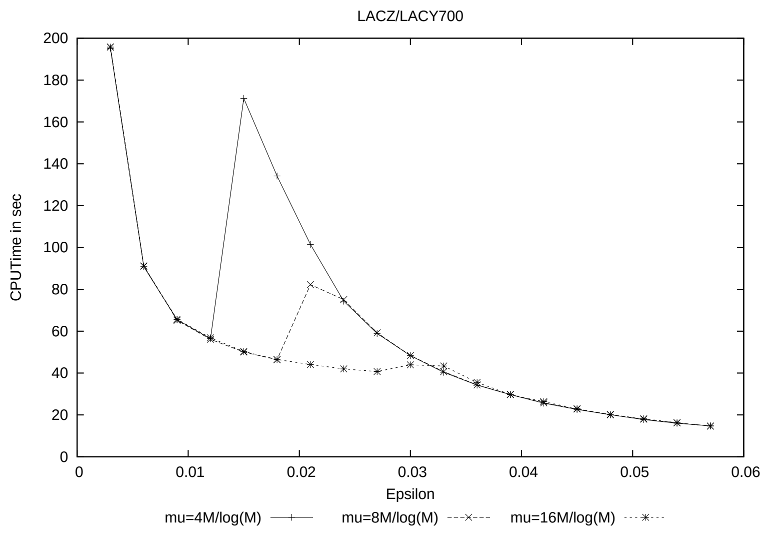


Figure 11.8: CPUTime after 10 independent simulations varying ϵ from 0.003 to 0.06 and μ in the set ($4M/\log(M)$, $8M/\log(M)$, $16M/\log(M)$) for LacZ/LacY model in the time interval [600,700].

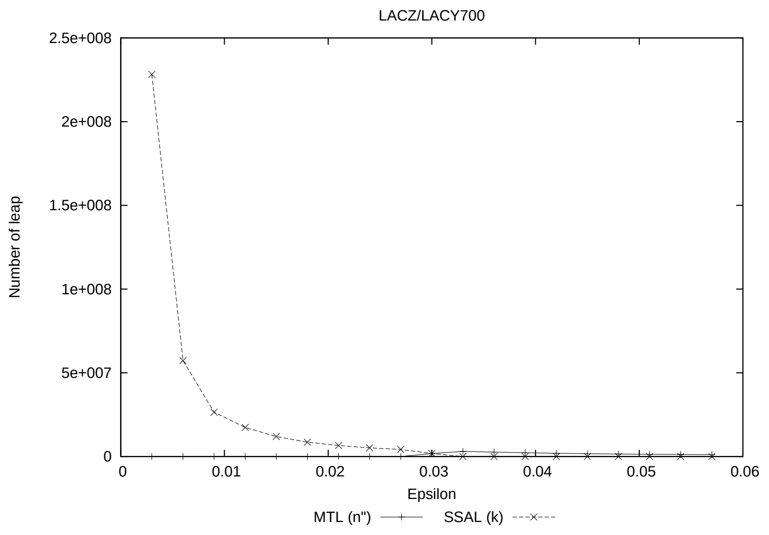


Figure 11.9: Number of leap executed with SSAL and MTL after 10 independent simulations varying ϵ from 0.003 to 0.06 and $\mu = 16M/\log(M)$ for LacZ/LacY model in the time interval [600,700].

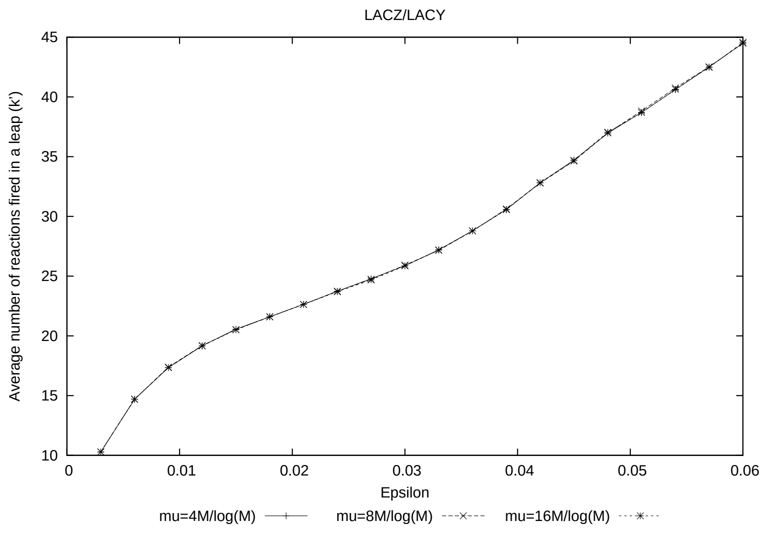


Figure 11.10: Average number of reactions fired in a leap after 1000 independent simulations varying ϵ from 0.003 to 0.06 and μ in the set $(4M/\log(M)$, $8M/\log(M)$, $16M/\log(M))$ for MAPK model.

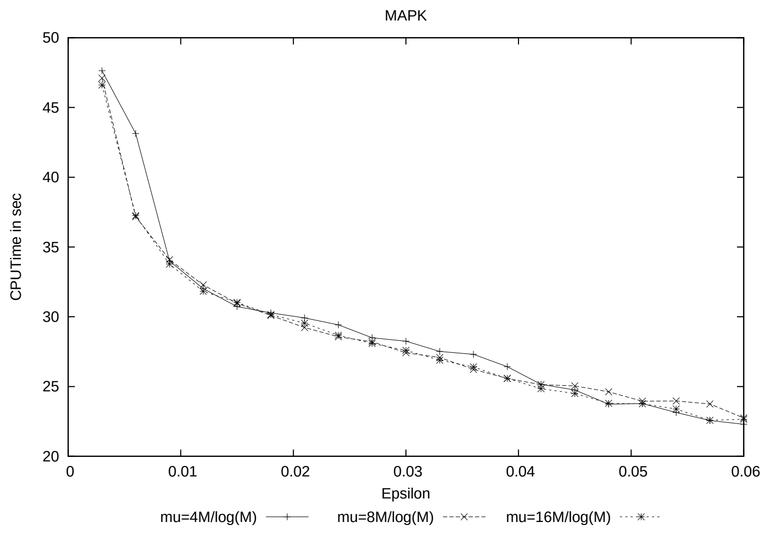


Figure 11.11: CPUTime after 1000 independent simulations varying ϵ from 0.003 to 0.06 and μ in the set $(4M/\log(M), 8M/\log(M), 16M/\log(M))$ for MAPK model.

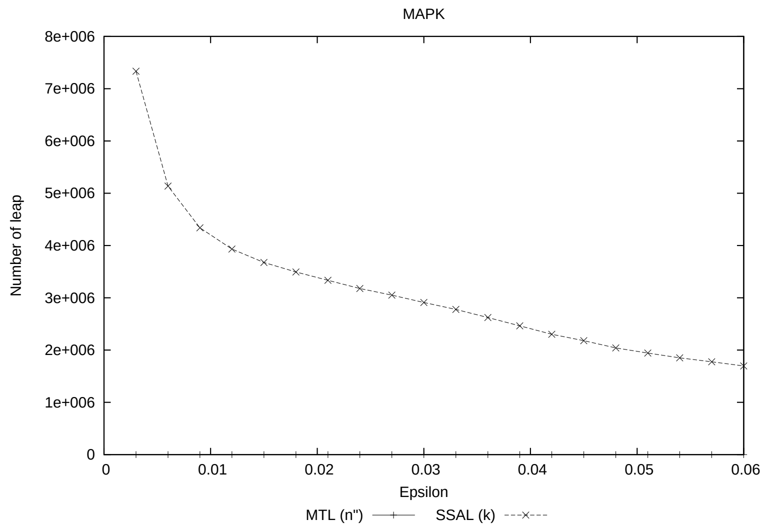


Figure 11.12: Number of leap executed with SSAL and MTL after 1000 independent simulations varying ϵ from 0.003 to 0.06 and $\mu = 16M/\log(M)$ for MAPK model.

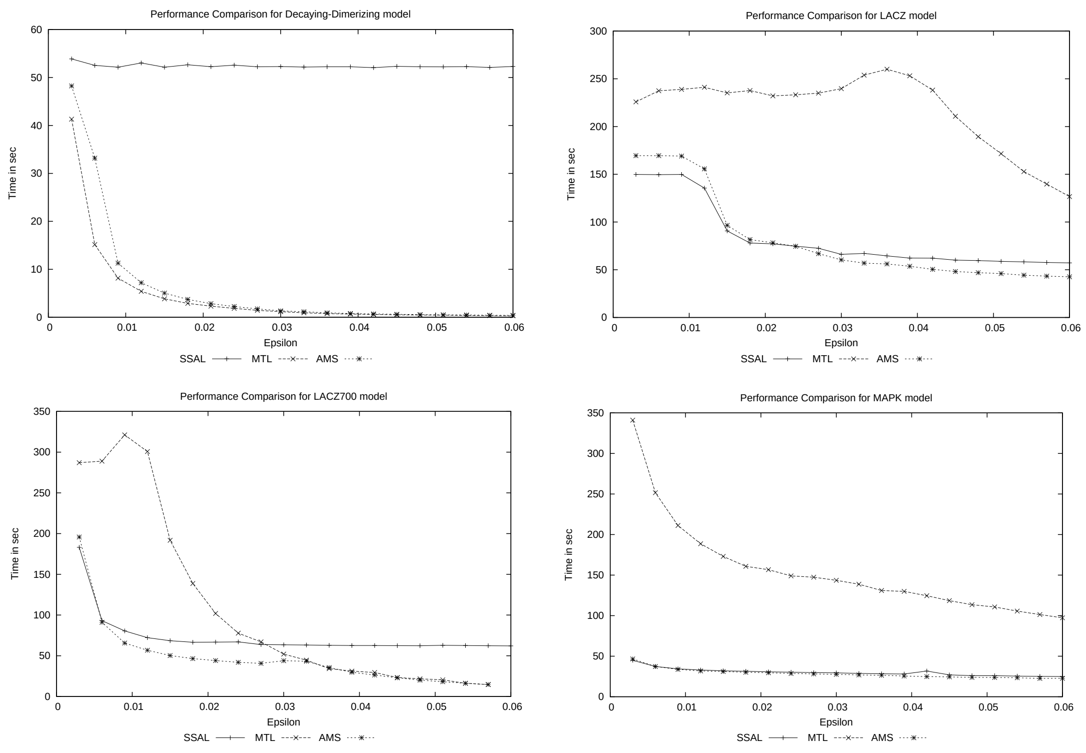


Figure 11.13: CPU Times after 1000 independent simulations for SSAL, MTL and AMS varying ϵ from 0.003 to 0.06 and considering $\mu = 16M/\log(M)$.

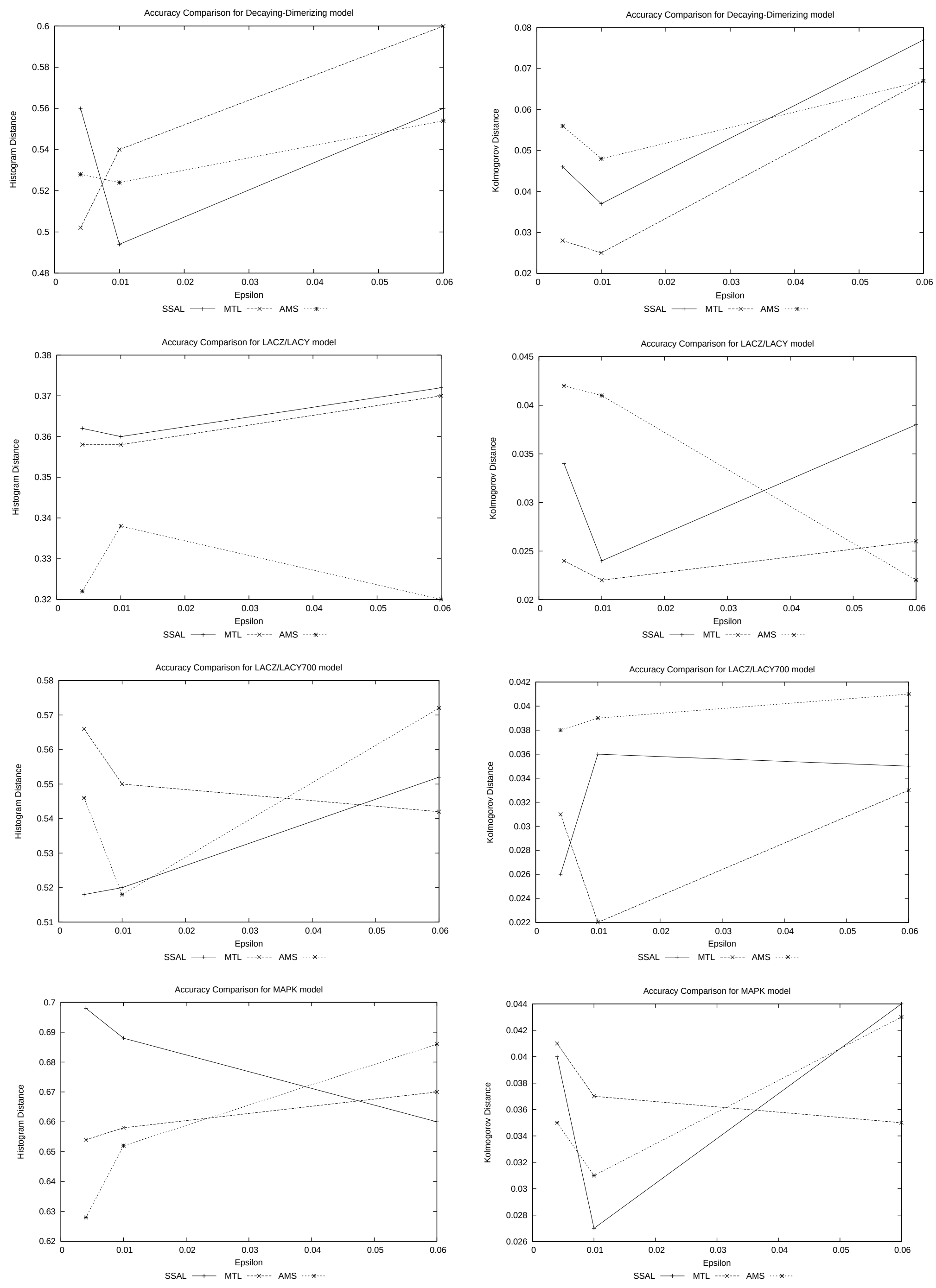


Figure 11.14: Accuracy for SSAL, MTL and AMS with $\mu = 16M/\log(M)$ for $\epsilon = 0.004$, $\epsilon = 0.01$ and $\epsilon = 0.06$ after 1000 independent simulations.

Part V

Ongoing work

Chapter 12

Ongoing Work

As we already discussed in Chap. 4 SSA is sequential in nature. This limits very much its parallelization across the method. However recently it is believed that spatiality can open new and efficient parallelized simulation methods. In our research we are investigating a method that merges ideas from space division and from τ-leaping. This method does not consider the diffusion of the molecules or molecular dynamics but it remains in the context of the classical stochastic formulation of chemical kinetics.

12.1 First Subvolume Direct Method

Suppose to have a system of `N` molecular species $\{S_1, \cdots, S_N\}$ interacting through `M` chemical reaction channels $\{R_1, \cdots, R_M\}$. Assume that the system is *well stirred*, in a constant volume `V` and in thermal (but not chemical) equilibrium. With the multivariate variable $X(t) = \{X_1(t), \cdots, X_N(t)\}$ as system state, where $X_i(t)$ is the number of molecules of species S_i in the system at time `t`. In this case, the quantity $a_j(x)dt$ gives the probability that one reaction R_j will occur somewhere in the volume in the next infinitesimal time interval $[t, t + dt)$. Suppose now to divide the volume V into a number N_{DIV} of *identical* sub-volumes. In this case, we obtain a partition P of identical sub-volumes $\{V_1, \cdots, V_{N_{DIV}}\}$. The term *identical* indicates that in each sub-volume the probability that a reaction occurs in one specific sub-volume in the next infinitesimal time dt is equal. In other words, potentially any molecule of any specie can react with the same probability in all sub-volumes. This means that we can define the propensity function of a reaction for a specific sub-volume V_m as the quantity $a_j^m(\mathbf{x})dt$ that gives the probability that a reaction R_j will occur in the sub-volume V_m in the next infinitesimal time interval $[t, t + dt)$. This new definition of propensity local to a specific sub-volume depends by a new definition of stochastic coefficient c_j^m. The new definitions of stochastic coefficient and propensity functions can be summarized as follows. If R_j is the unimolecular reaction $S_1 \xrightarrow{c_j} Product(s)$, if there are currently $\mathbf{x}_1$ molecules in the system, the probability that some one of them will undergo the R_j reaction in the sub-volume V_m in the next dt is $\mathbf{x}_1(c_j/(N_{DIV}))dt$. Here, c_j is rearranged becoming $c_j^m = (c_j/N_{DIV})$. The $c_j^m dt$ gives the probability that any particular molecule S_1 will so react in the next infinitesimal time dt in V_m. Thus the propensity function results $a_j^m(\mathbf{x}) = (c_j/(N_{DIV}))\mathbf{x}_1 = a_j/(N_{DIV})$. If R_j is a bimolecular reaction of the form $R_j : S_1 + S_2 \xrightarrow{c_j} Product(s)$, the probability that some one of the $\mathbf{x}_1$ and $\mathbf{x}_2$ S_1-S_2 pairs will react in V_m according to R_j in the next dt is therefore $\mathbf{x}_1\mathbf{x}_2(c_j/(N_{DIV}))dt$, where $(c_j/(N_{DIV}))dt$ gives the probability that a randomly chosen pair of S_1 and S_2 molecules will react in V_m according to R_j in the next infinitesimal time dt. In this case, the propensity function becomes $a_j^m(X(t)) = (c_j/(N_{DIV}))\mathbf{x}_1\mathbf{x}_2 = a_j/(N_{DIV})$. Finally, if the bimolecular reaction is $R_j : S_1 + S_1 \xrightarrow{c_j} Product(s)$, the propensity function now results $a_j^m(\mathbf{x}) = (c_j/(N_{DIV}))\frac{1}{2}\mathbf{x}_1(\mathbf{x}_1 - 1) = a_j/(N_{DIV})$, where $(c_j/(N_{DIV}))$ gives the probability that a randomly chosen two molecules of the same specie S_1 they will react in V_m according to R_j in the next infinitesimal time dt. Following the DM implementations of SSA for a sub-volume V_m we define a_0^m as follows.

$$a_0^m = \sum_{j=1}^{M} a_j^m. \tag{12.1}$$

The sum $\sum_{j=1}^{M} a_j^m d\tau$ gives the probability that one reaction in the reaction set will occur during the time $d\tau$ in the sub-volume V_m. Now according to the new definitions of propensity function a_j^m and the assumption that all sub-volumes are identical we obtain the mathematical equalities in Eq. 12.2, Eq. 12.3 and Eq. 12.4.

$$a_j = \sum_{m=1}^{N_{DIV}} \frac{a_j}{N_{DIV}} = \sum_{m=1}^{N_{DIV}} a_j^m \tag{12.2}$$

$$a_0 = \sum_{j=1}^{M} a_j = \sum_{j=1}^{M} \sum_{m=1}^{N_{DIV}} a_j^m = \sum_{m=1}^{N_{DIV}} \sum_{j=1}^{M} a_j^m = \sum_{m=1}^{N_{DIV}} a_0^m. \tag{12.3}$$

$$\frac{a_j}{a_0} = \frac{\frac{a_j}{N_{DIV}}}{\frac{a_0}{N_{DIV}}} = \frac{a_j^m}{a_0^m} = \frac{\sum_{m=1}^{N_{DIV}} a_j^m}{\sum_{m=1}^{N_{DIV}} a_0^m} \tag{12.4}$$

From the equality in Eq.12.4 we have that the point probability that a reaction R_j will occur in V is equal to the point probability that R_j will occur once we consider one of the sub-volumes. Similarly to DM and FRM we can define the probability $P(\tau, m, j)$ as follows.

$$P(\tau, m, j)d\tau \equiv \begin{array}{c}\text{probability at time t that the next reaction}\\ \text{will occur in the infinitesimal time interval}\\ (t+\tau, t+\tau+d\tau)\text{, it will be the reaction } R_j\\ \text{and it occurs in a specific sub-volume} V_m.\end{array} \tag{12.5}$$

It is also easy to see the following relation between the probability $P(\tau, j)$ seen for DM and FRM and $P(\tau, m, j)$.

$$P(\tau, j) = \sum_m P(\tau, m, j) \tag{12.6}$$

The equality in Eq. 12.6 can be read as follows. The probability $P(\tau, j)$ that the next reaction will occur somewhere in V and it will be the reaction R_j is the same of the probability that R_j will occur in one of the sub-volumes V_m. The probability theory allows to write $P(\tau, m, j)$ as a function of $P_1(j \mid \tau, m)$ and $P_2(\tau, m)$, where $P_1(j \mid \tau, m)$ gives the probability that the next reaction will be R_j, given that the next reaction occurs in some sub-volume V_m at time $(t+\tau, t+\tau+d\tau)$; and $P_2(\tau, m)$ gives the probability that the next reaction will occur in the interval $(t+\tau, t+\tau+d\tau)$ and it occurs inside the sub-volume with index m, independently on what that reaction will be.

$$P(\tau, m, j) = P_1(j \mid \tau, m) P_2(\tau, m). \tag{12.7}$$

The probability $P_1(j \mid \tau, m)$ that the next reaction will be R_j, given that the next reaction occurs in the sub-volume V_m at time $t + \tau$ is

$$P_1(j \mid \tau, m) = \frac{a_j^m}{a_0^m}. \tag{12.8}$$

Next, giving a sub-volume V_m the probability $P_2(\tau, m)$ that the next reaction will occur between times $t+\tau$ and $t+\tau+d\tau$ in a specific sub-volume V_m, independently on what that reaction will be results

$$P_2(\tau, m) = a_0^m exp(-\sum_m a_0^m \tau). \tag{12.9}$$

Here, $exp(-\sum_m a_0^m \tau)$ gives the probability that no reaction in any sub-volume V_m will occur during $(t, t+\tau)$ and then a reaction occur specifically in V_m. Substituting $P_1(j \mid \tau, m)$ and $P_2(\tau, m)$ in Eq. 12.7 and solving we obtain the following JPF:

$$P(\tau, m, j)d\tau = \frac{a_j^m}{a_0^m} a_0^m exp(-\sum_m a_0^m \tau) d\tau \tag{12.10}$$

$$= a_j^m exp(-\sum_m a_0^m \tau) d\tau \tag{12.11}$$

Again substituting $P(\tau, m, j)$ of Eq. 12.11 into Eq. 12.6 the result is

$$P(\tau, j)d\tau = \sum_m P(\tau, m, j)d\tau = \sum_m a_j^m exp(-\sum_m a_0^m \tau) d\tau. \tag{12.12}$$

Now, we introduce a Monte Carlo procedure to sample from the PDF in Eq. 12.12. We generate a tentative reaction time τ_m a sub-volume V_m according to the PDF $P_2(\tau, m)$ by drawing a uniformly distributed random number r_1 in $U(0, 1)$ and taking

$$\tau_m = \frac{1}{a_0^m(\mathbf{x})} \ln\left(\frac{1}{r_m}\right), \tag{12.13}$$

then we choose as next firing sub-volume that one which occurs first,

$$\tau = \text{ smallest } \tau_m \text{ for all } m = 1, \cdots, N_{DIV} \tag{12.14}$$

Moreover, because the value a_0^m is the same for all sub-volumes V^m, the formula τ_m in Eq. 12.13 changes only for the value of the number r_j, so

$$\min\{\tau_m \text{ for all } m = \{1, \cdots, N_{DIV}\}\} = \frac{1}{a_0^m} \ln\Big(\frac{1}{\max\{r_m\}}\Big). \tag{12.15}$$

As the minimum tentative τ_m is that one associated to the maximum uniform random number generated, it is not necessary to compute all tentative times τ_m.

Then by drawing another uniformly distributed random number r_2 and taking the index j for which

$$j = \text{the smallest integer satisfying } \sum_{j'=1}^{j} a_{j'}^m(\mathbf{x}) > r_2 a_0^m(\mathbf{x}), \tag{12.16}$$

we generate a sample index of the next reaction to fire according to $P_2(j \mid \tau, m)$ in Eq. 12.8.

We called the preceding procedure *First Sub-volume Direct Method* (FSDM). It a Monte Carlo procedure that iteratively generates τ and R_j according to Eq. 12.15 and Eq. 12.16. The name is inspired by the fact that it is an hybrid procedure. It samples the next reaction R_j and the waiting time τ using a DM-like procedure and the next sub-volume using a FRM-like procedure. As we already pointed out the main difference in the formulations of the FRM and the DM are the following. FRM generates independent tentative times for each reaction, and it chooses as next reaction that one which occurs first. Instead, DM generates independently only one tentative time and the reaction to fire. Our FSDM generates a tentative time for each sub-volume and it selects that one in which the next reaction occurs first. Than it generates the next reactions to fire exactly as DM. Schematically, FSDM makes the elementary steps summarized in Alg. 11. For $N_{DIV} = 1$ the operations processed by FSDM are the same of DM, in this case FSDM and DM have the same efficiency. Instead, for $N_{DIV} > 1$ at each step FSDM requires to generate and compare N_{DIV} random numbers. This last requirement makes FSDM less efficient than DM. But also more efficient of FRM for N_{DIV} smaller than M. The proof of exactness of the procedure is given in Appendix B.

12.1.1 Testing FSDM

We already proved the exactness of FSDM, here we want to show it in practice. We run a bunch of simulation upon a simple model known as Schlögl model [34]. This model is famous because executing a bunch of runs the final state population for the

Algorithm 11 First Sub-Volume Direct Method

```
m ← 1;
for j=1 to M do
    Compute $c_j/N_{DIV}$
end for
while $t < TIME$ do
    for j=1 to M do
        Compute $a_j^m(\mathbf{x})$
    end for
    Compute $a_0^m$
    for i=1 to $N_{DIV}$ do
        Generates $r_i$ in $U(0,1)$
    end for
    Determine the maximum $r_m$
    Generate $\tau$ according to Eq. 12.15
    Generates $r_2$ in $U(0,1)$ and generate $j$ according to Eq. 12.16
    $t \leftarrow t + \tau$; $\mathbf{x} \leftarrow \mathbf{x} + \nu_j$;
    print $(t, m, \mathbf{x})$
end while
```

specie S_2 figure out a bistable distribution. The Schlögl model as a two reversible coupled chemical reactions

$$S_1 + 2S_2 \underset{c_2}{\overset{c_1}{\rightleftharpoons}} 3S_2$$

$$S_3 \underset{c_4}{\overset{c_3}{\rightleftharpoons}} S_2.$$

The propensity functions are given by

$$a_1(\mathbf{x}) = \frac{c_1}{2}\mathbf{x}_1\mathbf{x}_2(\mathbf{x}_2 - 1),$$

$$a_2(\mathbf{x}) = \frac{c_2}{6}\mathbf{x}_2(\mathbf{x}_2 - 1)(\mathbf{x}_2 - 2),$$

$$a_3(\mathbf{x}) = c_3\mathbf{x}_3,$$

$$a_4(\mathbf{x}) = c_4\mathbf{x}_2$$

and the parameter values are:

$$c_1 = 3 \times 10^{-7}, c_2 = 10^{-4}, c_3 = 10^{-3}, c_4 = 3.5.$$

and initial state $X(t_0) = (\mathbf{x}_1 = 1 \times 10^5, \mathbf{x}_2 = 250, \mathbf{x}_3 = 2 \times 10^5)$.

Here, we measure the accuracy of our FSDM with respect to DM for the Schlögl model using distribution distances. Results are reported in Fig. 12.1. The kolmogorov self distance for the Schlögl model for 1000 samples has mean 0.079 and

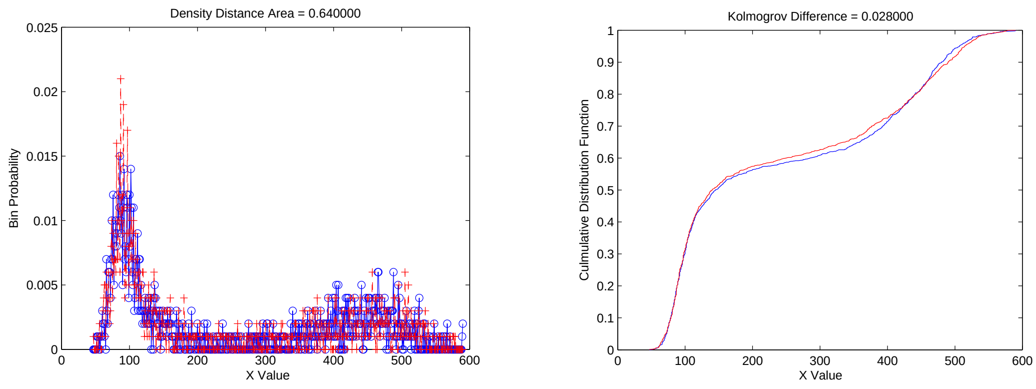


Figure 12.1: Accuracy for FSDM for $\epsilon = 0.03$ after 1000 independent simulations.

variance 0.49, whereas the Histogram self distance for the Schlögl model for 1000 samples and 544 bins has mean 0.83. The results in Fig. 12.1 show that FSDM is very close to the self distances, this confirms FSDM exactness as expected.

12.2 The Partitioning of the Volume

In this section we introduce a new method based on FSDM and SSAL. We called it *Partitioned SSALeaping* (PSSAL). The fundamental idea of PSSAL is the following. If a time interval τ' exists such that the leap condition holds, then each sub-volume is able to perform the sequence of reactions fired into the area assigned to it without any regards on the reactions occurring in the other sub-volumes. Then at the end of the leap the dynamics independently generated can be composed and the composition preserve accuracy of results. The practical application of this method involves the discussion of the following three issues. Why the leap condition produces independency, how to enforce the leap condition in a distributed environment and how to compose the dynamics. Our algorithm gives answers to these issues.

The first issue follows from the fact that enforcing the leap condition in a leap the relative changes in the propensity functions are related to the relative change in the molecular populations. Then if the propensity functions remain bounded in a time interval $[t, t + \tau')$ then the molecular populations do not change too much in τ', so the state $\mathbf{x}$ remains approximately bounded in τ'. Moreover, any reaction fired does not change too much the state in $[t, t + \tau')$. So the leap condition frees any sub-volume to take care of what the other sub-volumes are doing in τ'.

To enforce the leap condition in a distributed environment we assign a identical fraction of the tolerance to each sub-volumes. In other words, suppose to have a partition P with $N_{DIV} > 1$, we assign the tolerance ϵ_i / N_{DIV} to each sub-volume. We cannot assign more tolerance to a particular sub-volume because they have the same capability to react.

The explanation of the composition of the dynamics requires some further definitions. Given a time interval $[t, t + \tau')$ in which the leap condition holds we define the dynamics $\mathcal{D}_m$ of a sub-volume V_m as the set of pair $\{(t', j)$ sorted in increasing order of time $t' \in [t, t + \tau')\}$. Then given a partition $P = \{V_1, \cdots, V_{N_{DIV}}\}$ and the dynamics $\mathcal{D}_1, \cdots, \mathcal{D}_{N_{DIV}}$, we define the compositional operator $\oplus$ as the set of pair $\{(t', j)$ sorted in increasing order of time $t' \in [t, t + \tau')$ and $(t', j) \in \mathcal{D}_m$ for some $m = \{1, \cdots, N_{DIV}\}$. We denote the dynamics composition with $\mathcal{D}$, and we mathematically write it as follows.

$$\mathcal{D} = \oplus_m \mathcal{D}_m. \tag{12.17}$$

For example, consider the dynamics $\mathcal{D}_1 = \{(0.000001, 5), (0.000002, 3), (0.000004, 1), (0.000014, 9), (0.000027, 9)\}$ and the dynamics $\mathcal{D}_2 = \{(0.0000014, 7), (0.000003, 8), (0.00002, 1)\}$, the resulting dynamics composition $\mathcal{D}$ will be $\mathcal{D} = \mathcal{D}_1 \oplus \mathcal{D}_2 = \{(0.000001, 5), (0.0000014, 7), (0.000002, 3), (0.000003, 8), (0.000004, 1), (0.000014, 9), (0.00002, 1), (0.000027, 9)\}$. Note the order of the elements in $\mathcal{D}$.

12.2.1 Partitioned SSALeaping

The PSSAL performs the following simple steps. It starts creating the partition of the volume V. Each sub-volume executes SSAL until the leap condition is violated storing the sequence of reactions executed and the relative times. Regarding to the times τ'_m obtained by each sub-volume, PSSAL selects the minimum. This becomes the time τ' for this leap. Then for each sub-volume any reaction occurred after τ' are discarded because they do not contribute to the composition and the next state update.

Schematically, PSSAL takes M reactions and kinetic constants, N species, the number of sub-volumes N_{DIV}, one initial state $X(t_0)$, a stop time TIME and a tolerance parameter ϵ and performs the elementary steps summarized in Alg. 12. PSSAL calls the following SSAL procedure adapted to store the dynamics of a sub-volume. The choice of the minimum τ' comes from the fact that even though the sub-volumes are identical the fluctuations can generates very different dynamics and leap times τ'_m. Taking the minimum we expect that the dynamics composition derives considering the same time interval for all sub-volumes.

12.2.2 Testing PSSAL

To test the accuracy of PSSAL we realized our C language implementation of Alg. 12, and we apply it to simulate the Decaying-Dimerizing model seen in Chap. 10. We simulated the model by performing numerical tests for $N_{DIV} = 2$, $N_{DIV} = 4$ and $N_{DIV} = 8$. For each experiment we collected the final states of a selected species. We estimated the accuracy of the distribution distances between 1000 samples obtained by PSSAL and 1000 obtained by executing DM. Accuracy results are shown in

Algorithm 12 PSSALeaping(N_{DIV}, $\mathbf{x}$)

```
$m \leftarrow 1$;
$N_{DIV} \leftarrow number$
**for** j=1 to M **do**
  Compute $c_j/N_{DIV}$
**end for**
**while** $t < TIME$ **do**
  **for** j=1 to M **do**
    Compute $a_j^m(\mathbf{x})$
  **end for**
  Compute $a_0^m$
  $\tau \leftarrow 0$
  **for all** $m = 1$ to $N_{DIV}$ **do**
    $(D_m, \tau_m) \leftarrow$ execute SSALeaping according to $Alg.$ 13
  **end for**
  $mintau \leftarrow$ the smallest $\tau_m$
  $D \leftarrow \oplus_m \mathcal{D}_m$
  **for all** $p = 1$ to $D.end$ **do**
    **if** $p.\tau <= mintau$ **then**
      $\mathbf{x} \leftarrow \mathbf{x} + \nu_{p.j}$
    **end if**
  **end for**
  $t \leftarrow t + mintau$
  **print** $(t, \mathbf{x})$
**end while**
```

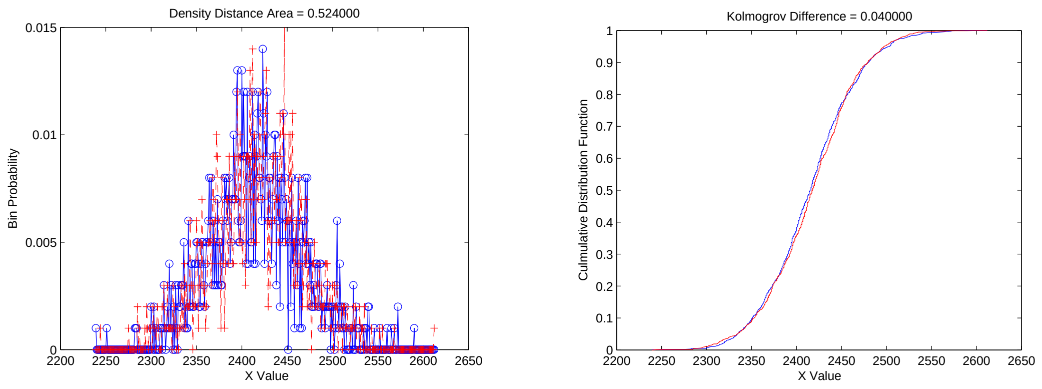


Figure 12.2: Accuracy for PSSAL for $\epsilon = 0.03$ after 1000 independent simulations and $N_{DIV} = 2$.

Algorithm 13 SSALeaping

```
$k \leftarrow 0$
$OK \leftarrow true$
$\tau' \leftarrow 0$
**while** $OK = true$ **do**
    Generate $r_1$ and $r_2$ in $U(0,1)$ and generate values for $\tau$ according to Eq. 5.6.
    Through binary search find $j$ according to Eq. 5.7.
    $\mathbf{x}' \leftarrow \mathbf{x}' + \nu_j$;
    **for all** $i \in ((Reactants(R_j) \cup Products(R_j)) \cap I_{rs})$ **do**
        **if** $\mid \mathbf{x}'_i - \mathbf{x}_i \mid > \epsilon_i \mathbf{x}_i / N_{DIV}$ **then**
            $OK \leftarrow false$
        **end if**
    **end for**
    $\tau' \leftarrow \tau' + \tau$
    $List \leftarrow List + (\tau', R_j)$
**end while**
return $(\tau', List)$
```

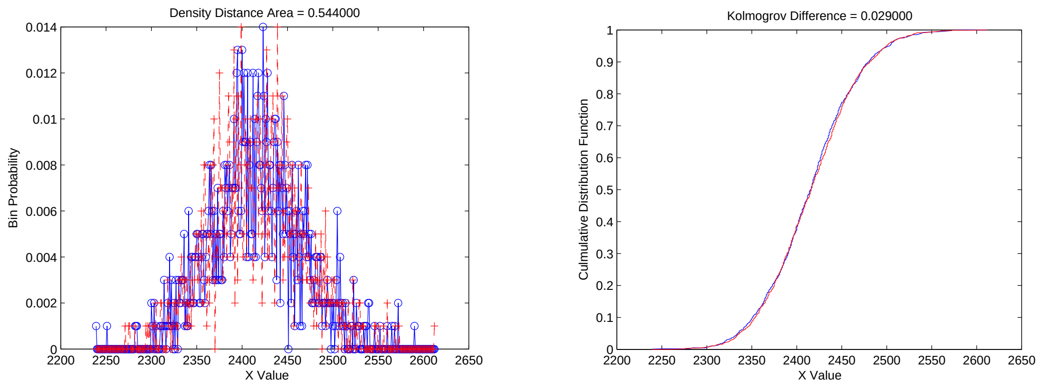


Figure 12.3: Accuracy for PSSAL for $\epsilon = 0.03$ after 1000 independent simulations and $N_{DIV} = 4$.

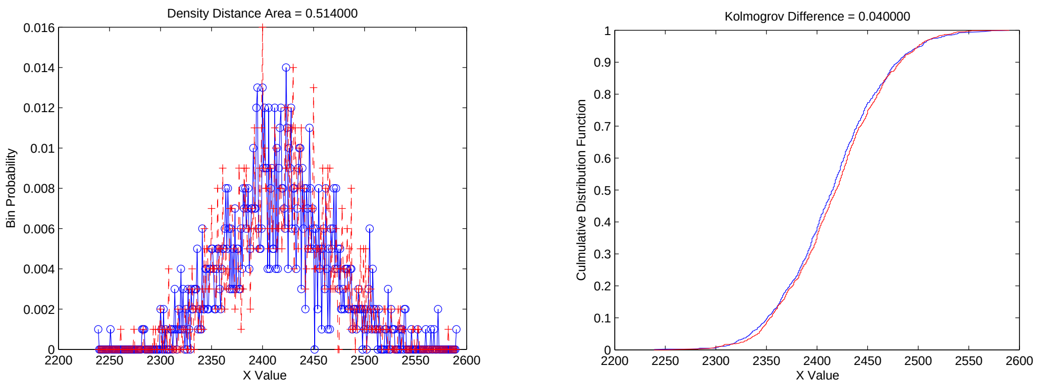


Figure 12.4: Accuracy for PSSAL for $\epsilon = 0.03$ after 1000 independent simulations and $N_{DIV} = 8$.

Fig. 12.2, Fig. 12.3 and Fig. 12.4. However, as we can see from Fig. 12.5, for larger value of N_{DIV} the average number of reactions fired in a leap substantially decreases. We justify the phenomenon simply stating that, due to the intrinsic randomness,

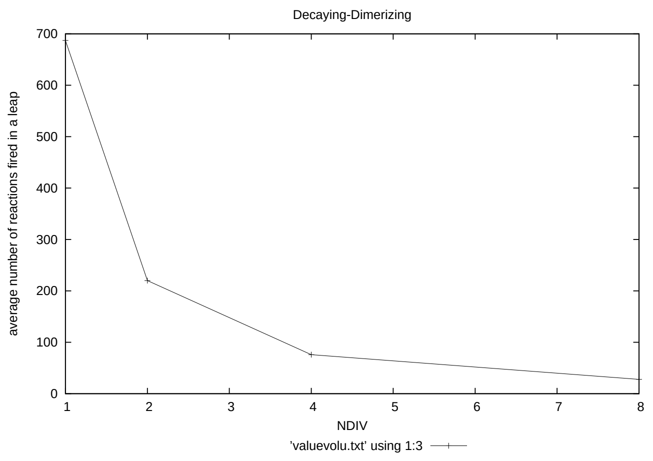


Figure 12.5: Average number of reactions fired in a leap changing N_{DIV}

reactions occur not homogeneously inside the volume. So for larger N_{DIV} this non homogeneity reduces further the number of reactions fired in a leap.

12.3 Conclusions

In this chapter we proposed *FSDM*, a new exact formulation of SSA based on the partitioning of the volume. We proved the equivalence with the DM formulation, and we have given a simple numerical test to show the accuracy of our method upon a realistic model. Then we introduced an ongoing method called Partitioned SSALeaping that is based on FSDM and SSALeaping. The main feature we pointed out is that in a leap the dynamics of a biological model can be obtained by composing the dynamics of the sub-volumes of the partitioned space. For now we executed the method upon a simple model test, we showed that the method accurately matched the results of DM, augmenting the number of sub-volumes as well. This confirmed that our method works maintaining the same accuracy of SSAL. However, we note that for partitions with a larger number of sub-volumes the number of reactions fired in a leap decreases. As one of the main advantages of a τ-leaping is that the propensity functions are computed a limited number of times when leap are long, we need to look for a way to build leap as long as possible. For this reason we let this issue for a future work. Additionally, in our discussions we completely neglected any reference to possible optimizations or efficiency. This is because this work is still in progress and much work have to be done. Apart this we consider the volume partitioning a very promising topic because independency and composition enable the parallelization of the computation of the dynamics of the sub-volumes.

A possible future application of our FSDM can be into hybrid simulation methods that consider the diffusion processes. As in general the methods that integrate the diffusion of the molecules divides the molecules among the sub-volumes even though the molecules diffuse very quickly. It is notorious that many of these methods spend much time to simulate the transition among neighbors sub-volumes. The volume partitioning can be integrated with these techniques to simulate quickly the molecules with very fast diffusion. We expect that this application accelerates the execution of these methods because many of the transitions can be saved. Please consider the preceding as possible applications but until now we have no specific proposals.

Part VI

Concluding Remarks and Future Work

Conclusions

In this thesis we mainly addressed problems about the efficient stochastic simulation of realistic biological models and the simulation of metabolic experiments as a powerful validation tool. In Chap. 8 we showed as the Systems Biology approach can provide a cheap, fast and powerful method for validating models proposed in literature. In the present case, we specified the model of SRI photocycle proposed by Hoff *et al.* [4] in a suitable developed simulator. This simulator was specifically designed to reproduce unprecedented *in silico* wet-lab experiments performed on metabolic networks with several possible controls exerted on them by the operator. Thanks to this, we proved that the screened model is able to explain correctly many light responses but unfortunately it was unable to explain some critical experiments, due to some unresolvable time scale problems. This also confirms that our simulator is useful to simulate metabolic experiments and interested reader can download our tool at the `http://sourceforge.net/projects/gillespie-qdc`.

This success calls for faster simulation techniques able to simulate realistic biological models. In order to accomplish this task in Chap. 9 we first investigated a parallel implementation of a devised version of FRM on the Graphic Processing Units. Differently from other methods in literature that perform parallel stochastic simulations on GPUs, our method implements what is known as the parallelism across the method. Numerical tests performed on a `GeForce 8600M GS` Graphic Card with 16 stream processors confirmed that simple general purpose applications are implementable using CUDA 2.0, however we identified some important limitations in the C language statements interpretation. These limitations constrain this technology only to simple general purpose applications. So the parallel implementation of stochastic algorithms of large size remains a challenge topic that is waiting for future advancements of GPUs technologies.

To achieve higher simulation speed we focused on well-known τ-leaping methods. We demonstrated that the most performant τ-leaping methods [163, 48, 131] are notoriously fast when leaps are long and many reactions can be fired. This is because the extra costs for pre-processing information necessary to advance the leap and avoid negative populations are more than compensated by the gain in simulation time due to the cumulative firing of a so great number of reactions. Unfortunately, we demonstrated that when leaps are short and few reactions can occur, those extra costs represent a burden for these methods and the performance can slow down. In particular, we highlighted that it is not the number of reactions fired *per se* that make such methods the most performant methods but it is the relation between the size of the model and the reactions firing that can make these method fast. In other words, in a leap to be fast these methods have to fire a number of reactions that exceeds the size of the model. Basing of the previous observations in Chap. 10 we proposed SSALeaping a new method in middle between DM and a τ-leaping. We proved experimentally upon realistic biological models that SSALeaping perform better than one of the most performant methods known as the Modified τ-leaping. Simply by generating each reaction as a conventional DM and checking the leap condition for that reaction this methods avoids to recompute the propensity functions at each step paying a cost proportional to the number of reactions effectively fired. For two of the three model tested SSALeaping performed better than MTL. Now we noted that already in the biological model tested the systems simulated can vary greatly in size and in the number of reactions fired in a leap, this makes very difficult in advance to establish what method will perform better. In our test we considered the algorithmic comparison with a method in the same class, but a more generalized comparison lacks. In fact, to the best of our knowledge do not exist scientific papers that compare simulation methods belonging to different classes. For instance, considering the methods proposed specifically to simulate stiff systems with τ-leaping and hybrid. Recently, adaptive methods [44] tried to put in the same framework solutions belonging to different simulation classes. Certainly these solution augment the number of model simulated efficiently but in our opinion, although these methods resumes the main advantages of the component algorithms they also suffer of their disadvantages.

Then in Chap. 11 we proposed *Adaptive Modified SSALeaping* (AMS) an adaptive method that exploiting the complementarity emerged between SSAL and MTL. During the simulation, our method switches between SSALeaping (SSAL) and Modified τ-leaping (MTL), according to conditions on the size of the model and the predicted length of the leap. The main issues related to the practical application of the proposed method are: the identification of the threshold that tells the algorithm to switch and the estimation of the number of reactions that will fire at each leap. We were able to solve both the issues. Numerical results performed on realistic biological models confirmed that AMS performs very well in all the cases tested. Moreover, maintaining the same accuracy of SSAL and MTL. Our AMS accelerated the execution of τ-leaping and and SSAL augmenting the number of models eligible to be simulated efficiently. Moreover, this has been made introducing negligible extra time. This prove that AMS is effective and promising to simulate realistic biological models. As our tests have not been applied specifically for stiff systems, for the future we will investigate efficient improvements of SSAL and AMS for stiff systems. In particular, we will investigate how propensity functions change in stiff systems. How the sequential generation of the reactions of our leap condition based methods can take into account those variations also when the simulation exhibit stiffness phenomenon. For the future we will also include our AMS into our QDC simulator letting the user the selection of the method to use for the simulation.

In Chap. 12 we also investigated other new parallelization techniques. We first presented a novel exact formulation of SSA based on the idea of partitioning the volume. We proved the equivalence between our method and DM, and we have given a simple numerical experiment to show its accuracy in practice. Then we proposed a variant of SSALeaping based on the partitioning of the volume, called Partitioned SSALeaping. The main feature we pointed out is that the dynamics of a system in a leap can be obtained by the composition of the dynamics processed by each sub-volume of the partition. This form of independency gives a different view with respect to existing methods. We only tested the method on a simple biological model, and we showed that the method accurately matched the results of DM, independently of the number of sub-volumes in the partition. This confirmed that the method works and that independency is effective. We have not already proposed a parallel implementation of this method because this work is still in progress and much work have to be done. Nevertheless, the Partitioned SSAleaping is a promising approach for a future parallelization on multi core (e.g. GPU's) or in many core (e.g. cluster) technologies. In order to obtain very efficient simulations with the parallelization remains to be solved some issue. First, how to compute the largest leap in this distributed environment. Second, how much reactions will be necessary to obtain real improvements against sequential τ-leaping methods.

Concluding in our opinion answer to the question what algorithm is best-suited for a given biochemical model represents today a great challenge in this field. Lack rigorous algorithmic analysis and strong experimentation. With SSALeaping providing an asymptotic complexity analysis we started making a step forward toward a more rigorous comparison among the simulation algorithms. However, we are far from a strong identification of the simulation conditions in which an algorithm is best than another. This is true both between algorithms in the same class and among algorithms belonging to different classes. Nevertheless, even though this question will receive answer we have to keep in mind that the simulation conditions and the realistic biological systems can be very different and can vary dynamically. So next generation algorithms must have the capability to adapt to such a variety. Developing such algorithms will be our focus.

Bibliography

[1] G. Battaglia, D. Cangelosi, R. Grossi and N. Pisanti. Masking patterns in sequences: A new class of motif discovery with don't cares. *Theor. Comput. Sci.*, 410(43):4327–4340, 2009.

[2] G. Cercignani, A. Frediani, S. Lucia and D. Petracchi. Competition-Integration of Blue and Orange Stimuli in Halobacterium salinarum Cannot Occur Solely in SRI Photoreceptor. *Biophysical Journal*, 79(3):1554–1560, 2000.

[3] G. Cercignani, S. Lucia, and D. Petracchi. Photoresponses of Halobacterium salinarum to Repetitive Pulse Stimuli. *Biophysical Journal*, 75(3):1466–1472, 1998.

[4] W.D. Hoff, K.H. Jung and J.L. Spudich. Molecular Mechanism of Photosignaling by Archaeal Sensory Rhodopsins. *Annual Reviews in Biophysics and Biomolecular Structure*, 26(1):223–258, 1997.

[5] S. Lucia, G. Cercignani, and D. Petracchi. Photosensory transduction in Halobacterium salinarium: evidence for a non-linear network of cross-talking pathways. *BBA-General Subjects*, 1334(1):5–8, 1997.

[6] A. Lupas and J. Stock. Phosphorylation of an N-terminal regulatory domain activates the CheB methylesterase in bacterial chemotaxis. *Journal of Biological Chemistry*, 264(29):17337–17342, 1989.

[7] W. Marwan, S.I. Bibikov, M. Montrone and D. Oesterhelt. Mechanism of Photosensory Adaptation inHalobacterium salinarium. *Journal of Molecular Biology*, 246(4):493–499, 1995.

[8] DA. McCain, LA. Amici, and JL. Spudich. Kinetically resolved states of the Halobacterium halobium flagellar motor switch and modulation of the switch by sensory rhodopsin I. *Journal of Bacteriology*, 169(10):4750–4758, 1987.

[9] V. Sourjik and H.C. Berg. Binding of the Escherichia coli response regulator CheY to its target measured in vivo by fluorescence resonance energy transfer. *Proceedings of the National Academy of Sciences*, 99(20):12669–12674, 2002.

[10] P.A. Spiro, J.S. Parkinson and H.G. Othmer. A model of excitation and adaptation in bacterial chemotaxis. *Proceedings of the National Academy of Sciences*, 94(14):7263–7268, 1997.

[11] JL. Spudich and RA. Bogomolni. Sensory rhodopsins of halobacteria. *Annual review of biophysics and biophysical chemistry*, 17(1):193–215, 1988.

[12] DA. McCain, LA. Amici, JL. Spudich. Kinetically resolved states of the Halobacterium halobium flagellar motor switch and modulation of the switch by sensory rhodopsin I. *Journal of bacteriology*, 169(10):4750, 1987.

[13] H. Kitano. Computational systems biology. *Nature*, 420(6912):206–210, 2002.

[14] H. de Jong. Modeling and simulation of genetic regulatory systems: a literature review. *Journal of Computational Biology*, 9(1):67–103, 2002.

[15] C. Eccher. Translation of Systems Biology Markup Language into Process Algebra. PhD thesis, International Doctorate School in Information and Communication Technologies DIT - University of Trento, 2006.

[16] M. Jirstrand H. Schmidt, and O. Wolkenhauer. Information technology in systems biology. *it-Information Technology,*, 48(3):133–139, 2006.

[17] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *CACM*, 31(9):1116–1127, 1988.

[18] D. F. Anderson. Incorporating postleap checks in tau-leaping. *The Journal of Chemical Physics*, 128(5):054103, 2008.

[19] David F. Anderson. A modified next reaction method for simulating chemical systems with time dependent propensities and delays. *The Journal of Chemical Physics*, 127(21):214107, 2007.

[20] A. Arkin, J. Ross and H. H. McAdams. Stochastic Kinetic Analysis of Developmental Pathway Bifurcation in Phage lambda-Infected Escherichia coli Cells. *Genetics*, 149(4):1633–1648, 1998.

[21] A. Auger, P. Chatelain and P. Koumoutsakos. R-leaping: Accelerating the stochastic simulation algorithm by reaction leaps. *The Journal of Chemical Physics*, 125(8):084103, 2006.

[22] F. Baras and M. M. Mansour. Reaction-diffusion master equation: A comparison with microscopic simulations. *Physical Review E*, 54(6):6139–6148, 1996.

[23] M. Barrio, K. Burrage, A. Leier and T. Tian. Oscillatory regulation of hes1: Discrete stochastic delay modelling and simulation. *PLoS Comput Biol*, 2(9):e117, 2006.

[24] G. Battaglia, R. Grossi, R. Marangoni and N. Pisanti. Mining biological sequences with masks. *Proceedings of the 20th International Workshop on Database and Expert Systems Application*, 193–197. IEEE, 2009.

[25] C. Berge. Hypergraphs: Combinatorics of Finite Sets. volume 45. Elsevier, 1989.

[26] David Bernstein. Simulating mesoscopic reaction-diffusion systems using the gillespie algorithm. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, 71(4):041103, 2005.

[27] Muffy Calder and Stephen Gilmore, editors. *Computational Methods in Systems Biology, International Conference, CMSB 2007, Edinburgh, Scotland, September 20-21, 2007, Proceedings*, volume 4695 of *Lecture Notes in Computer Science*. Springer, 2007.

[28] D. Bratsun, D. Volfson, Lev S. Tsimring and J. Hasty. Delay-induced stochastic oscillations in gene regulation. *Proceedings of the National Academy of Sciences of the United States of America*, 102(41):14593–14598, 2005.

[29] B. Brejová, D. Brown, and T. Vinar. Vector seeds: an extension to spaced seeds allows substantial improvements in sensitivity and specificity. In *WABI*, volume 2812 of *LNCS*, pages 39–54, Budapest, Hungary, 2003. Springer.

[30] A.E. Brockwell. Parallel Markov chain Monte Carlo simulation by pre-fetching. *Journal of Computational and Graphical Statistics*, 15(1):246–261, 2006.

[31] G. Stolting Brodal and R. Fagerberg. Cache oblivious distribution sweeping. In *Proc. 29th International Colloquium on Automata (ICALP), Languages, and Programming (ICALP)*, volume 2380 of *LNCS*, pages 426–438, Malaga, Spain, 2002. Springer.

[32] S. Burkhardt and J. Kärkkäinen. Better filtering with gapped q-grams. In *CPM*, volume 2089 of *LNCS*, pages 73–85, Jerusalem, Israel, 2001. Springer.

[33] Kevin Burrage, Shev Mac, and Tianhai Tian. Accelerated leap methods for simulating discrete stochastic chemical kinetics. pages 359–366. 2006.

[34] K. Burrage, S. Mac and T. Tian. Accelerated leap methods for simulating discrete stochastic chemical kinetics. volume 341 of *LNCS*, pages 359–366, 2006. Springer.

[35] T. Tian and K. Burrage. Poisson runge-kutta methods for chemical reaction systems. In *Proceedings of the Third International Workshop on Scientific Computing and Applications*, City University of Hong Kong, 2004.

[36] X. Cai. Exact stochastic simulation of coupled chemical reactions with delays. *The Journal of chemical physics*, 126(12):124108, 2007.

[37] X. Cai and Z. Xu. K-leap method for stochastic simulation of gene expression. *IEEE International Workshop on Genomic Signal Processing and Statistics, 2006. GENSIPS '06.* pages 81–82, 2006.

[38] Y. Cao, D. T. Gillespie, and L. R. Petzold. Accelerated stochastic simulation of the stiff enzyme-substrate reaction. *The Journal of chemical physics*, 123(14):144917, 2005.

[39] Y. Cao, D. T. Gillespie, and L. R. Petzold. Avoiding negative populations in explicit poisson tau-leaping. *The Journal of chemical physics*, 123(5):054104, 2005.

[40] Y. Cao, D. T. Gillespie, and L. R. Petzold. The slow-scale stochastic simulation algorithm. *The Journal of chemical physics*, 122(1):14116, 2005.

[41] Y. Cao, D. T. Gillespie, and L. R. Petzold. Efficient step size selection for the tau-leaping simulation method. *The Journal of chemical physics*, 124(4):44109, 2006.

[42] Y. Cao, H. Li, and L. Petzold. Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. *The Journal of chemical physics*, 121(9):4059–4067, 2004.

[43] Y. Cao and L. Petzold. Trapezoidal tau-leaping formula for the stochastic simulation of biochemical systems. in *Proceedings of Foundations of Systems Biology in Engineering*, FOSBE, pages 149–152, 2005.

[44] Y. Cao and L. Petzold. Slow-scale tau-leaping method. *Computer Methods in Applied Mechanics and Engineering*, 197(43-44):3472–3479, 2008.

[45] Y. Cao, D. Gillespie and L. Petzold. Multiscale stochastic simulation algorithm with stochastic partial equilibrium assumption for chemically reacting systems. *The Journal of chemical physics*, 206(2):395–411, 2005.

[46] Y. Cao, D. T. Gillespie and L. R. Petzold. Adaptive explicit-implicit tau-leaping method with automatic tau selection. *The Journal of Chemical Physics*, 126(22):224101, 2007.

[47] Y. Cao and L. Petzold. Accuracy limitations and the measurement of errors in the stochastic simulation of chemically reacting systems. *The Journal of Computational Physics*, 212(1):6–24, 2006.

[48] A. Chatterjee, D. G. Vlachos, and M. A. Katsoulakis. Binomial distribution based tau-leap accelerated stochastic simulation. *The Journal of Computational Physics*, 122(2):024112, 2005.

[49] A. Chatterjee. An overview of spatial microscopic and accelerated kinetic monte carlo methods. *Journal of Computer-Aided Materials Design*, 14:253–308(56), 2007.

[50] A. Chatterjee, K. Mayawala, J. S. Edwards and D. G. Vlachos. Time accelerated Monte Carlo simulations of biological networks using the binomial tau-leap method. *Bioinformatics*, 21(9):2136–2137, 2005.

[51] D. Chiarugi, M. Chinellato, P. Degano, G. Lo Brutto, and R. Marangoni. Feedbacks and oscillations in the virtual cell vice. *Lect. Not. Comp. Sci.*, 4210:93–107, 2006.

[52] Vincent Danos and Vincent Schächter, editors. *Computational Methods in Systems Biology, International Conference, CMSB 2004, Paris, France, May 26-28, 2004, Revised Selected Papers*, volume 3082 of *Lecture Notes in Computer Science*. Springer, 2005.

[53] K.P. Choi, F. Zeng, and L. Zhang. Good spaced seeds for homology search. *Bioinformatics*, 20(7):1053–1059, 2004.

[54] B. Chopard, L. Frachebourg, and M. Droz. Multiparticle Lattice Gas Automata for Reaction Diffusion Systems. *International Journal of Modern Physics C*, 5:47–63, 1994.

[55] C.J. Morton-Firth Predicting temporal fluctuations in an intracellular signalling pathway. *Journal of Theoretical Biology*, 192:117–128(12), 7 May 1998.

[56] NVIDIA Corporation. Nvidia geforce gtx 200 gpu architectural overview. Technical report, NVIDIA, May 2008.

[57] B. G. Cox. *Modern liquid phase kinetics / B.G. Cox.* Oxford University Press, Oxford ; New York :, 1994.

[58] L. Dematté, C. Priami, and A. Romanel. The beta workbench: a computational tool to study the dynamics of biological systems. *Briefings in Bioinformatics*, 9:437–449, 2008.

[59] P. K. Dhar, T. C. Meng, S. Somani, L. Ye, K. Sakharkar, A. Krishnan, A. B.M. Ridwan, S. Ho Kok Wah, M. Chitre and Zhu Hao. Grid Cellware: the first grid-enabled tool for modelling and simulating cellular processes. *Bioinformatics*, 21:1284–1287, 2005.

[60] J.-E. Duchesne, M. Giraud, and N. El-Mabrouk. Seed-based exclusion method for non-coding RNA gene search. In *COCOON*, volume 4598 of *LNCS*, pages 27–39, Banff, Canada, 2007. Springer.

[61] Weinan E, Di Liu, and E. Vanden-Eijnden. Nested stochastic simulation algorithm for chemical kinetic systems with disparate rates. *The Journal of Chemical Physics*, 123(19):194107, 2005.

[62] T. Eiter, K. Makino and G. Gottlob. Computational aspects of monotone dualization: A brief survey. *Discrete Applied Mathematics*, 156(11):2035–2049, 2008.

[63] J. Elf and M. Ehrenberg. Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases. *Systems Biology, IEE Proceedings*, 2:230–236, 2004.

[64] M. B. Elowitz, M. G. Surette, P. E. Wolf, J. B. Stock and S. Leibler. Protein mobility in the cytoplasm of escherichia coli. *J Bacteriol*, 181(1):197–203, January 1999.

[65] R. Eres, G.M. Landau and L. Parida. A combinatorial approach to automatic discovery of cluster-patterns. In *WABI*, volume 2812 of *LNCS*, pages 139–150, Budapest, Hungary, 2003. Springer.

[66] E. Eskin. From profiles to patterns and back again: a branch and bound algorithm for finding near optimal motif profiles. In *RECOMB '04: Proceedings of the eighth annual international conference on Resaerch in computational molecular biology*, pages 115–124, New York, NY, USA, 2004. ACM.

[67] James H Espenson. *Chemical kinetics and reaction mechanisms*. McGraw-Hill, New York, 2nd ed edition, 1995.

[68] S. Feng and E. Tillier. A fast and flexible approach to oligonucleotide probe design for genomes and gene families. *Bioinformatics*, 23(10):1195–1202, 2007.

[69] G. Franceschini. Proximity mergesort: optimal in-place sorting in the cache-oblivious model. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 291–299, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[70] M.L. Fredman and L. Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *J. Algorithms*, 21(3):618–628, 1996.

[71] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *40th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 285–297, New York, NY, USA, 1999.

[72] M. A. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. Phys. Chem. A*, 104:1876–1889, 2000.

[73] D. T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *The Journal of Chemical Physics*, 115(4):1716–1733, 2001.

[74] D. T. Gillespie and L. R. Petzold. Improved leap-size selection for accelerated stochastic simulation. *The Journal of Chemical Physics*, 119:8229–8234, 2003.

[75] D. T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *The Journal of Computational Physics*, 22(4):403–434, December 1976.

[76] Daniel T. Gillespie. Stochastic simulation of chemical kinetics. *Annual Review of Physical Chemistry*, 58(1):35–55, 2007.

[77] D.T. Gillespie. A rigorous derivation of the chemical master equation. *Physica A: Statistical Mechanics and its Applications*, 188(1-3):404–425, 1992.

[78] P. Godfrey, R. Shipley and J. Gryz. Algorithms and analyses for maximal vector computation. *VLDB Journal: Very Large Data Bases*, 16(1):5–28, October 2006.

[79] J. Goutsias. Quasiequilibrium approximation of fast reaction kinetics in stochastic biochemical systems. *The Journal of Chemical Physics*, 122(18):184102, 2005.

[80] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. Sewak Sharma. Discovering all most specific sentences. *ACM Trans. Database Syst.*, 28(2):140–174, 2003.

[81] D. Gunopulos, H. Mannila, R. Khardon and H. Toivonen. Data mining, hypergraph transversals, and machine learning (extended abstract). In *PODS '97: Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 209–216, New York, NY, USA, 1997. ACM.

[82] H. Kuthan Self-organisation and orderly processes by individual protein complexes in the bacterial cell. *Progress in Biophysics and Molecular Biology*, 75:1–17(17), 2001.

[83] Li H. and Petzold L. Logarithmic direct method for discrete and stochastic simulation of chemically reacting systems. Technical report, 2006.

[84] J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann, 2001.

[85] Leonard A. Harris, Aaron M. Piccirilli, Emily R. Majusiak, and Paulette Clancy. Quantifying stochastic effects in biochemical reaction networks using partitioned leaping. *Physical Review E*, 79:051906, 2009.

[86] E. L. Haseltine and J. B. Rawlings. On the origins of approximations for stochastic chemical kinetics. *The Journal of chemical physics*, 123(16):164115, October 2005.

[87] Eric L. Haseltine and James B. Rawlings. Approximate simulation of coupled fast and slow reactions for stochastic chemical kinetics. *The Journal of Chemical Physics*, 117(15):6959–6969, 2002.

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

[89] Martin Howard and Andrew D. Rutenberg. Pattern formation inside bacteria: fluctuations due to low copy number of proteins. *Physical Review Letters*, 90:128102, 2003.

[90] Y. Hu and T. Li. Highly accurate tau-leaping methods with random corrections. *The Journal of chemical physics*, 130:124109, 2009.

[91] L. Ilie and S. Ilie. Fast computation of good multiple spaced seeds. In *WABI*, volume 4645 of *LNBI*, pages 346–358, Philadelphia, PA, USA, 2007. Springer.

[92] M. Jeschke, A. Park, R. Ewald, R. Fujimoto, and A.M. Uhrmacher. Parallel and distributed spatial simulation of chemical reactions. *Principles of Advanced and Distributed Simulation, 2008. PADS '08. 22nd Workshop on*, pages 51–59, 2008.

[93] Richard M. Karp, Raymond E. Miller, and Arnold L. Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. In *STOC '72: Proceedings of the fourth annual ACM symposium on Theory of computing*, pages 125–136, New York, NY, USA, 1972. ACM Press.

[94] D.J. Kavvadias and E.C. Stavropoulos. An efficient algorithm for the transversal hypergraph generation. *J. Graph Algorithms and Applications*, 9(2):239–264, 2005.

[95] U. Keich, M. Li, B. Ma, and J. Tromp. On spaced seeds for similarity search. *Discr. Appl. Math.*, 138(3):253–263, 2004.

[96] L. Khachiyan, E. Boros, K. Elbassioni, and V. Gurvich. An efficient implementation of a quasi-polynomial algorithm for generating hypergraph transversals and its application in joint generation. *Discr. Appl. Math.*, 154(16):2350–2372, 2006.

[97] Thomas R. Kiehl, Robert M. Mattheyses, and Melvin K. Simmons. Hybrid simulation of cellular behavior. *Bioinformatics*, 20(3):316–322, 2004.

[98] A.M. Kierzek. Stocks: Stochastic kinetic simulations of biochemical systems with gillespie algorithm. *BIOINFORMATICS*, 18:470–481, 2002.

[99] A. M. Kierzek, J. Zaim, and P. Zielenkiewicz. The Effect of Transcription and Translation Initiation Frequencies on the Stochastic Fluctuations in Prokaryotic Gene Expression. *Journal of Biological Chemistry*, 276(11):8165–8172, 2001.

[100] H. Kitano. Systems biology: a brief overview. *Science*, 295(5560):1662–1664, 2002.

[101] G. Kucherov, L. Noé, and M. A. Roytberg. Subset seed automaton. In *CIAA*, volume 4783 of *LNCS*, pages 180–191, Praque, Czech Republic, 2007. Springer.

[102] G. Kucherov, L. Noé, and M.A. Roytberg. Multiseed lossless filtration. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 2(1):51–61, 2005.

[103] G. Kucherov, L. Noé, and M.A. Roytberg. A unifying framework for seed sensitivity and its application to subset seeds. *J. Bioinform. and Comput. Biology*, 4(2):553–570, 2006.

[104] H. Kurata, H. El-Samad, T.-M. Yi, M. Khammash, and J. Doyle. Feedback regulation of the heat shock response in e. coli. *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on*, 1:837–842, 2001.

[105] H. Li, Y. Cao, L.R. Petzold, and D.T. Gillespie. Algorithms and software for stochastic simulation of biochemical reacting systems. *Biotechnology progress*, 24(1):56, 2008.

[106] H. Li and L. Petzold. Efficient parallellization of stochastic simulation algorithm for chemically reacting systems on the graphics processing unit. Technical report, Dept. Computer Science, University of California, Santa Barbara, 2007.

[107] H. Li and L.R. Petzold. Stochastic simulation of biochemical systems on the graphics processing unit, 2007.

[108] M. Li, B. Ma, D. Kisman, and J. Tromp. Patternhunter ii: Highly sensitive and fast homology search. *J. Bioinform. and Comput. Biology*, 2(3):417–440, 2004.

[109] T. Li. Analysis of explicit tau-leaping schemes for simulating chemically reacting systems. *ANALYSIS*, 6(2):417–436.

[110] Z. Liu and Y. Cao. Detailed comparison between stochsim and ssa. *Systems Biology, IET*, 2(5):334–341, 2008.

[111] L. Lok and R. Brent. Automatic generation of cellular reaction networks with moleculizer 1.0. *Nature Biotechnology*, 23(1):131–136, 2005.

[112] K. Luby-Phelps, P. E. Castle, D. L. Taylor and F. Lanni. Hindered diffusion of inert tracer particles in the cytoplasm of mouse 3t3 cells. *Proceedings of the National Academy of Sciences of the United States of America*, 84(14):4910–4913, 1987.

[113] T. T. Marquez-Lago and K. Burrage. Binomial tau-leap spatial stochastic simulation algorithm for applications in chemical kinetics. *The Journal of Chemical Physics*, 127(10):104101, 2007.

[114] E. A. Mastny, E. L. Haseltine and J. B. Rawlings. Two classes of quasi-steady-state model reductions for stochastic kinetics. *The Journal of Chemical Physics*, 127(9):094106, 2007.

[115] M. Matsumoto and T. Nishimura. Dynamic creation of pseudorandom number generators. In *Monte-Carlo and Quasi-Monte Carlo Methodsproceedings of a conference held at the Claremont Graduate University, Claremont, USA, June 22-26, 1998*, pages 56–69, Springer, 1998.

[116] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998.

[117] H. h. Mcadams and A. Arkin. Stochastic mechanisms in gene expression. *PNAS*, 94(3):814–819, February 1997.

[118] James M. Mccollum, Gregory D. Peterson, Chris D. Cox, Michael L. Simpson, and Nagiza F. Samatova. The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior. *Computational Biology and Chemistry*, 30(1):39–49, February 2006.

[119] T. C. Meng, S. Somani, and P. Dhar. Modeling and simulation of biological systems with stochasticity. *In Silico Biol*, 4(3):293–309, 2004.

[120] M. Michael, F. Nicolas, and E. Ukkonen. On the complexity of finding gapped motifs. *Journal of Discrete Algorithms*, Elsevier, 2008.

[121] A. P. Minton. Macromolecular crowding and molecular recognition. *Journal of Molecular Recognition*, 6(4):211–214, 1993.

[122] C. J. Morton-Firth and D. Bray. Predicting temporal fluctuations in an intracellular signalling pathway. *J Theor Biol*, 192(1):117–128, May 1998.

[123] C. J. Morton-Firth, T. S. Shimizu and D. Bray. A free-energy-based stochastic simulation of the tar receptor complex. *Journal of Molecular Biology*, 286(4):1059 – 1074, 1999.

[124] I. Mura. Exactness and Approximation of the Stochastic Simulation Algorithm. Tecnical Report, COSBI, Italy, 2008.

[125] NVIDIA Corporation. *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*, 2.0 edition, 2008.

[126] O. Givon. Extracting macroscopic dynamics: model problems and algorithms. *Nonlinearity*, 17:55–127(1), 2004.

[127] J. Pahle. *Stochastic simulation and analysis of biochemical networks.* PhD thesis.

[128] L. Parida, I. Rigoutsos, A. Floratos, D. Platt and Yuan Gao. Pattern discovery on character sets and real-valued data: linear bound on irredundant motifs and an efficient polynomial time algorithm. In *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 297–308, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[129] X. Peng, W. Zhou and Y. Wang. Efficient binomial leap method for simulating chemical kinetics. *The Journal of Chemical Physics*, 126(22):224109, 2007.

[130] M. F. Pettigrew and H. Resat. Modeling signal transduction networks: A comparison of two stochastic kinetic simulation algorithms. *The Journal of chemical physics*, 123(11):114707, 2005.

[131] M. F. Pettigrew and H. Resat. Multinomial tau-leaping method for stochastic kinetic simulations. *The Journal of Chemical Physics*, 126(8):084101, 2007.

[132] L. Petzold and H. Li. Efficient parallelization of stochastic simulation algorithm for chemically reacting systems on the graphics processing unit. *International Journal of High Performance Computing Applications*, 24(2):107–116, 2010.

[133] A. Phillips, L. Cardelli, and Castagna G. A graphical representation for biological processes in the stochastic π-calculus. *Transactions in Computational Systems Biology*, 4230:123–152, 2006.

[134] Branislav Rovan and Peter Vojtás, editors. *Mathematical Foundations of Computer Science 2003, 28th International Symposium, MFCS 2003, Bratislava, Slovakia, August 25-29, 2003, Proceedings*, volume 2747 of *Lecture Notes in Computer Science*. Springer, 2003.

[135] S. Ramsey, D. Orrell, and H. Bolouri. Dizzy: Stochastic simulation of large-scale genetics regulatory networks. *J. Bioinf. Comp. Biol.*, 3:415–436, 2005.

[136] C. V. Rao and A. P. Arkin. Stochastic chemical kinetics and the quasi-steady-state assumption: Application to the gillespie algorithm. *The Journal of Chemical Physics*, 118(11):4999–5010, 2003.

[137] M. Rathinam, L. R. Petzold, Y. Cao and D. T. Gillespie. Stiffness in stochastic chemically reacting systems: The implicit tau-leaping method. *The Journal of Chemical Physics*, 119(24):12784–12794, 2003.

[138] M. Rathinam, L. R. Petzold, Y. Cao and D. T. Gillespie. Consistency and stability of tau leaping schemes for chemical reaction systems. *SIAM Multiscale Modeling*, 4:2005, 2005.

[139] M. Rathinam and H. El Samad. Reversible-equivalent-monomolecular tau: A leaping method for "small number and stiff" stochastic chemical systems. *The Journal of Chemical Physics*, 224(2):897–923, 2007.

[140] S. Reinker, R.M. Altman, and J. Timmer. Parameter estimation in stochastic biochemical reactions. *Systems Biology, IEE Proceedings*, 153(4):168–178, 2006.

[141] A. B. M. Ridwan, A. Krishnan and P. Dhar. A parallel implementation of gillespie's direct method. *International Conference on Computational Science*, LNCS, vol. 3037, pp. 284291. Springer, 2004

[142] V. J. Rodriguez, J. A. Kaandorp, M. Dobrzynski and J. G. Blom. Spatial stochastic modelling of the phosphoenolpyruvate-dependent phosphotransferase (pts) pathway in escherichia coli. *Bioinformatics*, 22(15):1895–1901, 2006.

[143] A. Romanel and C. Priami. On the computational power of blenx. *Theor. Comp. Sci.*, 411:542–565, 2010.

[144] M. R. Roussel and R. Zhu. Validation of an algorithm for delay stochastic simulation of transcription and translation in prokaryotic gene expression. *Physical Biology*, 3(4):274, 2006.

[145] H. Salis and Y. Kaznessis. Accurate hybrid stochastic simulation of a system of coupled chemical or biochemical reactions. *The Journal of Chemical Physics*, 122(5):054103, 2005.

[146] A. Samant and D. G. Vlachos. Overcoming stiffness in stochastic simulation stemming from partial equilibrium: A multiscale monte carlo algorithm. *The Journal of Chemical Physics*, 123(14):144114, 2005.

[147] A. Samant, B. Ogunnaike, and Dionisios Vlachos. A hybrid multiscale monte carlo algorithm (hymsmc) to cope with disparity in time scales and species populations in intracellular networks. *BMC Bioinformatics*, 8(1):175, 2007.

[148] M S Samoilov and A P Arkin. Deviant effects in molecular reaction pathways. *Nat Biotechnol*, 24(10):1235–1240, October 2006.

[149] Bethe A. Scalettar, James R. Abney, and Charles R. Hackenbrock. Dynamics, structure, and function are coupled in the mitochondrial matrix. *Proceedings of the National Academy of Sciences of the United States of America*, 88(18):8057–8061, 1991.

[150] F. W. Schneider. Review: [untitled]. *SIAM Review*, 42(2):361–363, 2000.

[151] S. Schnell and T. E. Turner. Reaction kinetics in intracellular environments with macromolecular crowding: simulations and rate laws. *Prog Biophys Mol Biol*, 85(2-3):235–260, 2004.

[152] B. Schoeberl, C. Eichler-Jonsson, E. D. Gilles, and G. Müller. Computational modeling of the dynamics of the map kinase cascade activated by surface and internalized egf receptors. *Nat Biotechnol*, 20(4):370–375, 2002.

[153] T. P. Schulze. Kinetic Monte Carlo simulations with minimal searching. *Physical Review E*, 65(3):036704, March 2002.

[154] M. Schwehm, U. Brinkschulte, KE Grosspietsch, C. Hochberger, and EW Mayr. Parallel stochastic simulation of whole-cell models. In *Proceedings of International Conference on Architecture of Computing Systems. ARCS*, volume 2002, 2002.

[155] T. S. Shimizu and D. Bray. *Computational cell biology - the stochastic approach*, chapter 10. MIT Press, Cambridge, MA, 2001.

[156] Audrius B. Stundzia and Charles J. Lumsden. Stochastic simulation of coupled reaction-diffusion processes. *Journal of Computational Physics*, 127(1):196 – 207, 1996.

[157] Y. Sun and J. Buhler. Designing multiple simultaneous seeds for dna similarity search. *J. Comput. Biology*, 12(6):847–861, 2005.

[158] Y. Sun and J. Buhler. Designing patterns for profile hmm search. *Bioinformatics*, 23(2):42–43, 2007.

[159] Zoltan Szallasi, Jörg Stelling, and Vipul Periwal. *System Modeling in Cellular Biology: From Concepts to Nuts and Bolts.* The MIT Press, 2006.

[160] Kouichi Takahashi, Satya Nanda Vel Arjunan, and Masaru Tomita. Space in systems biology of signaling pathways - towards intracellular molecular crowding in silico. *FEBS Letters*, 579(8):1783 – 1788, 2005. Systems Biology.

[161] B. Teusink, B. M. Bakker, and H. V. Westerhoff. Control of frequency and amplitudes is shared by all enzymes in three models for yeast glycolytic oscillations. *Biochim. Biophys. Acta*, 1275:204–212, 1996.

[162] R. Thar and M Kuhl. Bacteria are not too small for spatial sensing of chemical gradients: An experimental evidence. *Proceedings of the National Academy of Sciences of the United States of America*, 100(10):5748–5753, 2003.

[163] T. Tian and K. Burrage. Binomial leap methods for simulating stochastic chemical kinetics. *The Journal of Chemical Physics*, 121(21):10356–10364, 2004.

[164] T. Tian and K. Burrage. Parallel implementation of stochastic simulation for large-scale cellular processes. In *HPCASIA '05: Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region*, page 621, Washington, DC, USA, 2005. IEEE Computer Society.

[165] A. M. Turing. The Chemical Basis of Morphogenesis. *Royal Society of London Philosophical Transactions Series B*, 237:37–72, 1952.

[166] T. E. Turner, S. Schnell, and K. Burrage. Stochastic approaches for modelling in vivo reactions. *Computational Biology and Chemistry*, 28(3):165–178, July 2004.

[167] E. Ukkonen. Structural analysis of gapped motifs of a string. In *MFCS*, volume 4708 of *LNCS*, pages 681–690, Ceský Krumlov, Czech Republic, 2007. Springer.

[168] A. S. Verkman. Solute and macromolecule diffusion in cellular aqueous compartments. *Trends Biochem Sci*, 27(1):27–33, January 2002.

[169] Jörg R. Weimar. Cellular automata approaches to enzymatic reaction networks. In *ACRI '01: Proceedings of the 5th International Conference on Cellular Automata for Research and Industry*, pages 294–303, London, UK, 2002. Springer-Verlag.

[170] P. O. Westermark and A. Lansner. A model of phosphofructokinase and glycolytic oscillations in the pancreatic β-cell. *Biophys. J.*, 85:126–139, 2003.

[171] Darren J. Wilkinson. *Stochastic Modelling for Systems Biology (Mathematical and Computational Biology).* Chapman & Hall/CRC, April 2006.

[172] M. Yoshimi, Y. Osana, Y. Iwaoka, A. Funahashi, N. Hiroi, Y. Shibata, N. Iwanaga, H. Kitano, and H. Amano. The design of scalable stochastic biochemical simulator on fpga. *2005 IEEE nternational Conference on Field-Programmable Technology, 2005. Proceedings.* , pages 339–340, 2005.

[173] Y. Chushak, B. Foy, and J. Frazier. Stochastic simulations of cellular biological processes., In *HPCMP-UGC '07: Proceedings of the 2007 DoD High Performance Computing Modernization Program Users Group Conference*, pages 425–429, 2007.

[174] L. A. Harris and P. Clancy. A partitioned leaping approach for multiscale modeling of chemicalreaction dynamics., *The Journal of Chemical Physics*, 125:144107, 2006.

[175] J. M. Mccollum, G. D. Peterson, C. D. Cox, M. L. Simpson, and N. F. Samatova. The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior., *Computational Biology and Chemistry*, 30:39–49, 2006.

[176] NVIDIA Corporation. *The CUDA Compiler Driver NVCC*, 2.0 edition, April 2008.

[177] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware., *Computer Graphics Forum*, 26(1):80–113, 2007.

[178] Cristian Dittamo and Davide Cangelosi. Optimized parallel implementation of gillespie's first reaction method on graphics processing units. In *ICCMS*, pages 156–161, 2009.

[179] D. Gillespie. *Formal Methods for Computational Systems Biology*, volume 5016/2008, chapter Simulation Methods in Systems Biology, pages 125–167. Springer Berlin / Heidelberg, May 2008.

[180] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

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

[182] D. Cangelosi. SSALeaping: Efficient Leap Condition Based Direct Method Variant for the Stochastic Simulation of Chemical Reacting System. *Accepted Paper to the International ICTS Conference on Simulation Tools and Techniques*, Torremolinos, Spain , 2010.

[183] R. C. H. Cheng and G. M. Feast. Gamma variate generators with increased shape parameter range. *Commun. ACM*, 23(7):389–395, 1980.

[184] A. Slepoy, A. P. Thompson and S. J. Plimpton A constant-time kinetic Monte Carlo algorithm for simulation of large biochemical reaction networks. *The Journal of Chemical Physics*, 128(20):205101, 2008.

[185] F. Bonchi. Saturated Semantics for Reactive Systems. *Twenty First Annual IEEESymposium on Logic in Computer Science (LICS06)*, Seattle, 12-08-2006.

Appendix A

QDC's Syntax

More formally, a QDC program is made of mandatory and optional declarations. Mandatory are the declarations of the species, the volume, the reactions set, the initial amounts for the species. Whereas variables and events can be omitted. In this case simulation does not have perturbations. Any QDC program is an ASCII file. The user can declare any *name* (sequence of Letters [a-zA-Z] and Digits [0-9]). A name is a generic identifier, here we will write speciesname or variablename in order to distinguish the names given to species by that given to the variables. We will write naturalnumber, realnumber, baserate, and time in order to distinguish the *numbers* given for the different cases in which a specific number type must be provided. However, because QDC uses the symbols $NULL, volume, >, \$,, +, , , 0, 1$ as language statements, these symbols cannot be introduced in the declarations.
The syntax of the QDC language has the following syntax:

$$
\begin{array}{lcl}
QDCProgram & := & ProgramWithEvents \\
 & | & ProgramWithoutEvents \\
 & & \\
ProgramWithEvents & := & SpeciesList \\
 & & VolumeDec \\
 & & ReactionsList \\
 & & StateVector \\
 & & VariablesList \\
 & & EventList \\
 & & \\
ProgramWithoutEvents & := & SpeciesList \\
 & & VolumeDec \\
 & & ReactionsList \\
 & & StateVector \\
 & & \\
SpeciesList & := & speciesname \\
 & | & speciesname, SpeciesList
\end{array}
$$

$$
\begin{array}{lll}
VolumeDec & := & volume, realvalue \\
 & & \\
ReactionsList & := & Reaction \\
 & | & Reaction \\
 & & ReactionsList \\
 & & \\
Reaction & := & baserate, Reactants > Products \\
 & | & \$variablename, Reactants > Products \\
 & | & \_, Instantaneous > Instantaneous \\
 & & \\
Reactants & := & speciesname \\
 & | & speciesname + speciesname \\
 & | & 2speciesname \\
 & | & NULL \\
 & & \\
Products & := & speciesname \\
 & | & speciesname + speciesname \\
 & | & 2speciesname \\
 & | & NULL \\
 & & \\
Instantaneous & := & naturalnumberspeciesname \\
 & | & naturalnumberspeciesname + Instantaneous \\
 & & \\
StateVector & := & SpeciesAmount \\
 & | & SpeciesAmount \\
 & & StateVector \\
 & & \\
SpeciesAmount & := & speciesname, time, naturalnumber \\
 & & \\
VariablesList & := & Variable \\
 & | & Variable, VariablesList \\
 & & \\
Variable & := & \$variablename \\
 & & \\
EventsList & := & Event \\
 & | & Event \\
 & & EventsList \\
 & & \\
Event & := & 0, \$variablename, realnumber \\
 & | & 1, \$variablename, realnumber
\end{array}
$$

The syntax expressed before define in a very simple way any correct QDC pro-

gram. Below we provide also how our Tool maps any syntactically well formed QDC file into the input parameters of the Gillespie's Direct Method. To do that we define the *state of a SSA simulation* as the tuple $Z = (\mathbf{X}(t), E, S, R, C, A, t)$, where $X(t) = \mathbf{x} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ is the state of the system at a given time t, $E = \{e_1, \cdots, e_h\}$ is the set of events, $S = \{S_1, \cdots, S_N\}$ is the set of all the species name, $R = \{R_1, \cdots, R_M\}$ is the set of all the system reactions, $C = \{c_1, \cdots, c_M\}$ is the set of all the stochastic coefficients, $A = \{a_1(\mathbf{x}), \cdots, a_M(\mathbf{x})\}$ is the set of the propensity values of the reactions and t is the simulation time.
Given $Z = (\mathbf{X}(t), E, S, R, C, A, t)$, an instance $S_1, S_2, \cdots, S_N$ obtained by the *Species-List* production is mapped into the set of species names $S = \{S_1, \cdots, S_N\}$. The volume declaration in the *VolumeDec* production becomes the volume of the compartment expressed in Liter. For example, $volume, 0.0000000000015$ is interpreted as the declaration of the volume identifier $V = 0.0000000000015$. A sequence of $M = 8$ reaction declarations, as follows:

- $r1, S_1 > S_2$
- $r2, S_1 + S_2 > S_3$
- $r3, 2S_2 > S_4 + S_5$
- $r4, NULL > S_2$
- $\$var1, S_1 > S_2$
- $\$var2, S_1 + S_2 > S_3$
- $\$var3, 2S_2 > S_4 + S_5$
- $\$var4, NULL > S_2$

is interpreted as the reaction set: $R = \{\ S_1 \xrightarrow{r1} S_2,\ S_1 + S_2 \xrightarrow{r2} S_3,\ 2S_2 \xrightarrow{r3} S_3,\ NULL \xrightarrow{r4} S_2,\ S_1 \xrightarrow{\$var1}, S_2,\ S_1 + S_2 \xrightarrow{\$var2} S_6,\ 2S_2 \xrightarrow{\$var3} S_3,\ NULL \xrightarrow{\$var4} S_2\}$. The initial value of the variables $\$var1, \$var2, \$var3, \$var4$ must be assigned using the statement $0, \$s, r$ in the *EventList* production. Anyway, our tool interprets the values $r_1, r_2, r_3, r_4, \$var1, \$var2, \$var3, \$var4$ as deterministic reaction rates $k_1, \cdots, k_8$, then it converts them according by the conversion Table 7.1 and generates the stochastic coefficients set $C = \{c_1, \cdots, c_8\}$. Once the conversion is completed, our tool computes the propensity function set $A = \{a_1(\mathbf{x}), \cdots, a_8(\mathbf{x})\}$ using the classical propensity functions seen at the beginning of this Appendix.
Events declaration requires a different treatement. In fact, events are external statements to the SSA formulation. They reprensent perturbation actions with effects on the system evolution. The action fired by the occurrence of an event is subject to the verification of a trigger condition. The trigger condition can dependent both on the simualtion time and on stoichiometric conditions. In any case, the list of

events declared using the *EventsList* production form the set E of the events and in the following we will give how our tool executes the events and what effects on the state of the simulation they induce. To do this we first define the *State Transition Function* $\mapsto$. *State Transition Function* $\mapsto$ is a function that maps a given tupla into another that is the result of a possible evolution of the first one. In the Gillespie's Stochastic Simulation Algorihtm we can see $\mapsto$ as the function that starting from a state of the simulation $Z = (\mathbf{X}(t), E, S, R, C, A, t)$ and given the pair (τ, j) passes to a new state $Z'' = (\mathbf{X}(t+\tau), E, S, R, C, A', t)$ where $\mathbf{X}(t+\tau) = \mathbf{X}(t) + \nu_j$ and A' is the new set of propensity function that changed their value for effect of the changing of the molecular amount. More formally

$$\frac{(\tau, j) \leftarrow GillespieTauandJselectionStep}{< Z > \mapsto < (\mathbf{X}[(\mathbf{X}(t) + \nu_j)/\mathbf{X}(t)], E, S, R, C, A', t) >} \tag{A.1}$$

We write $A[r/\$s]$ to specify that in a given set A we substitute the value of the variable $\$s$ with a new value r. We will also write $A/\$s$ to indicate the removal of the element with identifier $\$s$ from the set A.

To complete the definition of the *State Transition Function* $\mapsto$ we give a set of Rules to define how our tool executes Events.
Let's suppose now to have an *instantaneous reaction* that we call *inst* for short, defined in the following way, $\_, l_1 s1 + l_2 s2 > l_3 s3$ where $l_1, l_2, l_3 > 0$ are stoichiometric values. When the system can satisfy the stoichiometry of the left side of the equation, QDC immediately puts the simulation in pause and it remains in pause since the *inst* trigger condition holds. To represent the pause of the simulation we write $(\mathbf{X}(t), \mathtt{E}, \mathtt{S}, \mathtt{R}, \mathtt{C}, \mathtt{A}, \mathtt{t})$ calling this new state Z'. Since the simulation is in pause all elements in Z' cannot be modified except $X(t)$. To deal with *inst* we identified four distinct cases. The first one happens when the simulation run and at a certain point the system has enough molecules to trigger this event. Formally, the Rule A.2 expresses this case.

$$\frac{\mathbf{x}_{s1} \geq l_1 \ \mathbf{x}_{s2} \geq l_2 \ inst \in E}{< Z > \mapsto < Z'[(\mathbf{X}(t) + \nu_{inst})/\mathbf{X}(t)] >} \tag{A.2}$$

The second case Rule A.3 takes place when the simulation has been already paused and the system has enough molecules to trigger again *inst*. Note that *inst* is not removed by any elements of Z'.

$$\frac{\mathbf{x}_{s1} \geq l_1 \ \mathbf{x}_{s2} \geq l_2 \ inst \in E}{< Z' > \mapsto < Z'[(\mathbf{X}(t) + \nu_{inst})/\mathbf{X}(t)] >} \tag{A.3}$$

The Rule A.4 considers the third case, when *inst* cannot occur anymore. In this case, the state of the simulation is restored to a value in which all elements of Z' are no longer in pause and the simulation can continue normally.

$$\frac{\mathbf{x}_{s1} < l_1 \text{ or } \mathbf{x}_{s2} < l_2 \ inst \in E}{< Z' > \mapsto < Z >} \tag{A.4}$$

Last, the Rule A.5 considers the fourth case in which the event cannot be triggered because there are not enough molecules and the simulation continues normally.

$$\frac{\mathbf{x}_{s1} < l_1 \text{ or } \mathbf{x}_{s2} < l_2 \;\; inst \in E}{< Z > \mapsto < Z >} \tag{A.5}$$

Supposing to have the statement s, t', n, called *aug* for short, the Rule A.6 expresses the following behavior: if $t' \geq t$, only two elements of Z change due to *aug*: the state $\mathbf{x}_s$ of the species with name s, augments its current value of n molecules and the event set E, deleting *aug*, decreases its size.

$$\frac{t' \geq t \;\; aug \in E}{< Z > \mapsto < (\mathbf{X}[(\mathbf{x}_s + n)/\mathbf{x}_s], E/aug, S, R, C, A, t) >} \tag{A.6}$$

Supposing now to have the statement $1, t', \$s, r$, called *chan* for short, the Rule A.7 expresses the following behavior: if $t' \geq t$, due to *chan*, Z the current value of the stochastic coefficient with name $\$s$ in C is substituted with the new value $f(r)$ and the event set E, deleting *chan*, decreases its size. Here, $f(r)$ represents the deterministic reaction rate r opportunely converted using Table 7.1. Consequently the reaction in which $\$s$ appears as kinetic parameter must be recomputed.

$$\frac{t' \geq t \;\; chan \in E}{< Z > \mapsto < (\mathbf{X}(t), E/chan, S, R, C[r/\$s], A[a_j[r/\$d](\mathbf{x})/a_j(\mathbf{x})], t) >} \tag{A.7}$$

Appendix B

Proof of Exactness of FSDM

To prove the equivalence between FRM and DM Gillespie showed in [75] that both procedures sample the random pair (τ,j) according to the same PDF. Here we want to do the same between FSDM and DM.
Let $\bar{P}(\tau, j)d\tau$ be the probability that the procedure described in Eq. 12.14 and Eq. 12.16 will result in the next reaction being R_j and occurring in the time interval $(t + \tau, t + \tau + d\tau)$.

From Eq. 12.14 and Eq. 12.16 we may write

$$\bar{P}(\tau, j) = \sum_m Prob\{\tau < \tau_{m,j} < \tau + d\tau\} Prob\{\tau_\nu > \tau, \text{ all } \nu \neq m\}, \tag{B.1}$$

where, the probability $\bar{P}(\tau, j)d\tau$ is equal to the probability that the next reaction is just R_j, that occurs in one of the sub-volume $m = 1, \cdots, N_{DIV}$ with tentative time τ_m and any other reaction in any other sub-volume occurs after τ_m.

From Eq. 12.8 and Eq. 12.9 we see that the first factor is just

$$\begin{aligned} Prob\{\tau < \tau_{m,j} < \tau + d\tau\} &= Prob\{\tau < \tau_m < \tau + d\tau\} \cdot \\ &\quad \cdot Prob\{next = R_j\} \\ &= exp(-a_0^m \tau) a_j^m d\tau. \end{aligned} \tag{B.2}$$

For the second factor in Eq. B.1, Gillespie proved in [75] that

$$Prob\{\tau_\nu > \tau, \text{ all } \nu \neq m\} = \prod_{\substack{\nu=1 \\ \nu \neq m}}^{N_{DIV}} exp(-a_0^\nu \tau) d\tau. \tag{B.3}$$

Inserting the equation in Eq. B.2 and Eq. B.3 in Eq. B.1 we have

$$\bar{P}(\tau, j) = (\sum_m a_j^m) exp(-\sum_m a_0^m \tau). \tag{B.4}$$

From Eq. 12.2 we know that a $\sum_m a_j^m = a_j$ and from Eq. 12.3 we know that $\sum_m a_0^m = a_0$. Substituting them into Eq. B.4 we have

$$\bar{P}(\tau, j) = a_j \cdot exp(-a_0 \tau) = P^{DM}(\tau, j) \tag{B.5}$$

Thus, the First Subvolume Direct Method, as defined in Eq. 12.14 and Eq. 12.16 is as rigorous and exact as the Direct Method.