UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

Facoltà di Scienze MM.FF.NN.
Dottorato di Ricerca in Informatica
XXII Ciclo

Stochastic algorithms for biochemical processes

Paolo Cazzaniga

Supervisor Prof. Giancarlo Mauri
Tutor Prof. Paola Bonizzoni
PhD Coordinator Prof. Stefania Bandini

ANNO ACCADEMICO 2008–2009

Acknowledgments

I would like to express my sincere gratitude to Professor Giancarlo Mauri who has been my supervisor. He gave me many helpful suggestions and important advice during the course of my PhD.

Special thanks are due to Daniela Besozzi for her constant encouragement, her useful help (on many occasions) and her friendship.

I would also like to thank Dario Pescini for his assistance with writing code, his great effort to explain things and for all the last minute rushes to airports and train stations because we were always late.

A particular thanks to Prof. Stephen Gilmore and Jane Hillston who hosted me for 6 months in Edinburgh. And I should also say thank you to my flatmates, the guys working at the LFCS and all the other people I met there.

I wish to thank the people working at DISCo for their friendship, help and for the parties we had.

I am grateful to the people who helped me during my PhD, and to my friends for the fun and laughter we shared.

Indeed, I owe a special thanks to my family for its support during this period and to Laura who believed in me and supported me throughout my PhD.

Contents

Introduction

Aims and motivations

After the completion of the human genome sequencing (and of a lot of other genomes), the main challenge for the modern biology is to understand complex biological processes such as metabolic pathways, gene regulatory networks and cell signalling pathways, which are the basis of the functioning of living cells. This goal can only be achieved by using mathematical modelling tools and computer simulation techniques, to integrate experimental data and to make predictions on the system behaviour that will be then experimentally checked, so as to gain insights into the working and the general principles of organization of biological systems.

In fact, the formal modelling of biological systems allows the development of simulators, which can be used to understand how the described system behaves in normal conditions, and how it reacts to (simulated) changes in the environment or to alterations of some of its components. Simulations present many advantages over conventional experimental biology in terms of cost, ease to use and speed. For instance, some 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. Therefore, the development of predictive *in silico* models offers opportunities for unprecedented control over the system.

In the last few years a wide variety of models of cellular processes have been proposed, based on different formalisms. For example, chemical kinetic models attempt to represent a cellular process as a system of distinct chemical reactions. In this case, the network state is defined by the instantaneous quantity (or concentration) of each molecular species of interest in the cell, and different molecular species may interact via one or more reactions. Usually, reactions are represented by a system of coupled differential equations that relate the quantity of reactants to the quantity of products, according to a reaction rate and other parameters.

Recently, it has been pointed out that transcription, translation and other cellular processes may not behave deterministically but instead are

better modelled as random events [123]. Several models address this concern by abandoning differential equations in favour of stochastic relations that describe each chemical reaction in terms of molecular collisions [6, 74].

The use of stochastic methods for the study of biological systems is motivated by the fact that these systems are usually composed by many chemical interactions among a large number of chemical species, whereby the molecular quantities involved can be small (few tens of molecules). In systems having these characteristics, *noise* plays a major role in the system's dynamics [128].

Two different kind of noise can be identified in biological systems: *extrinsic* and *intrinsic* [59, 176]. The first one is related to the experimental conditions; for instance, the variation of temperature, pressure, light, or fluctuations of other cellular factors, are all sources of extrinsic noise. On the other hand, there are stochastic events occurring during the processes of gene expression, at the level of transcription, translation and protein degradation, which result in intrinsic noise.

The role of noise in biological systems has been studied and quantified [59, 176, 192], proving that the classic deterministic and continuous approach (like, for instance, ordinary differential equations) is unsuitable for the modelling, simulation and analysis of phenomena like cellular pathways, especially when the gene transcription and translation machinery is involved.

The deterministic approach is based on the *law of mass action*, an empirical law which provides a simple relation between reaction rates and molecular species concentrations. Given the initial molecular concentrations, by using the law of mass action, a temporal description of the component concentrations can be obtained. The law of mass action considers chemical reactions to be macroscopic, continuous and deterministic. However, in the study of "small" systems, the law of mass action becomes inadequate, and it more is suitable to apply stochastic approaches, since (1) they take into account the discreteness of the quantity of the molecular species and the inherently random character of the phenomena; (2) they are in accordance with the theories of thermodynamics and stochastic processes; and (3) they are appropriate for the description of systems characterised by instability phenomena [198].

On the other hand, one major problem related to stochastic methods is that they are difficult to implement analytically; hence, they are implemented by means of numerical simulations whose computation time is usually very expensive.

In this thesis we provide a discrete and stochastic framework for the modelling, simulation and analysis of biological and chemical systems. To this aim, we will start from the description of other techniques and methods that are present in literature, as the basis to build and compare our approach.

At a different level of abstraction many formalisms have been employed

to model biological systems. Some of these have been originally developed by computer scientists to model systems of interacting components, such as Petri Nets [165] or π-calculus [133], while others have been proposed for the study of biochemical systems, like κ-calculus [43] and Bio-PEPA [36]. Moreover, formalisms such as membrane systems [154], originally proposed as a model of computation inspired by biology, have recently found application to the formal description and modelling of biological phenomena.

Starting from the notion of membrane systems (or P systems), we provide the definition of one of their particular variant called *dynamical probabilistic P systems* (DPPs) [162]. P systems, and in particular DPPs, represent an appropriate tool for the modelling of biochemical systems, they provide a suitable structure (called membrane structure) which can be used to describe the spatial arrangement of the compartments involved in a system. Moreover, inside each compartment, a set of chemical reactions written as multiset rewriting rules can be specified, together with a set of molecular species specified as multisets of objects. The evolution of a system is obtained by means of the application of the rules on the objects currently present inside the membranes. In the basic definition of P systems, the rules are applied in a nondeterministic and maximally parallel manner, and at each step all the objects which can evolve should evolve. In DPPs, the maximal parallelism has been mitigated by assigning probabilities to the rules, and these values vary according to the system state. By exploiting these values, it is possible to provide a description of the system's dynamics, that is, DPPs allow to reproduce the stochastic variations of the elements (i.e. chemical species) occurring in the system. However, this description is only *qualitative*, in the sense that an effective (physical) time streamline cannot be directly associated to the evolution steps of the system.

The temporal dynamics of a biochemical system composed by many volumes can be simulated by integrating a stochastic algorithm with the framework of DDPs. The *stochastic simulation algorithm* [74] represents the seminal procedure used for reproducing the exact dynamics of a system that is enclosed in a single volume, which is assumed to be well stirred (i.e. homogeneous, in the sense that molecules are considered uniformly distributed) and at constant temperature. One of the main drawbacks of this procedure is the computational time required to obtain the behaviour of a system, because this task is achieved by simulating one reaction per step. More recently, several algorithms have been proposed in order to overcome this limitation; among others, in this thesis we recall the *next reaction method* [72] – a procedure that executes reactions in a sequential manner, but exploits suitable data structure to update the system's state and to handle the additional information required during the simulation, thus speeding up the computation – and the *tau-leaping* algorithm [77], a method based on a strategy that allows to select and execute in parallel several reactions per step. Tau-leaping represents one of the most efficient algorithms for the

description of the dynamics of biochemical systems. However, these algorithms share the same limitation: they are only applicable to homogeneous systems enclosed in a single volume.

Nevertheless, many cellular processes are characterised by a spatial heterogeneity [182], where diffusion plays an important role for the system dynamics, e.g. the living cell. In order to describe the spatial heterogeneity, there exist methods that divide the reaction volume in a number of subvolumes ,and then consider both reaction and diffusion processes to describe the behaviour of the entire system.

For instance, the *next subvolume method* [57] provides the dynamics of a heterogeneous system composed by many subvolumes, by sequentially simulating a single reaction or diffusion event inside one subvolume selected at the beginning of each iteration. The computation time required to execute this procedure is usually high, hence, to speed up the computation the same data structures used in the next reaction method are exploited here to manage the information of the subvolumes. Another example of stochastic algorithm for the simulation of heterogeneous systems is the *binomial tau-leap spatial stochastic simulation algorithm* [117] which is based on the next subvolume method, but exploits a particular version of the tau-leaping algorithm to describe the dynamics of the subvolumes, resulting in a more efficient simulation procedure (with respect to the next subvolume method). The main drawback of these two procedures concerns the high computation time required to run a simulation, since both algorithms update the internal state of a single subvolume during each iteration.

In order to overcome the limitations of DPPs and of the stochastic algorithms cited here, in this thesis we introduce a novel method for the modelling and simulation of biochemical systems, which combines the descriptive power of DPPs with the efficiency of tau-leaping algorithm. This approach, called τ-DPP [30], exploits the membrane structure and the system definition of DPPs, with the aim of describing multiple volume systems, and uses a modified version of the tau-leaping algorithm for the description of the system behaviour. Differently from DPPs, where the obtained dynamics is only qualitative, with τ-DPP it is possible to provide the *quantitative* behaviour, by assigning a time increment to each simulation step. In order to investigate systems consisting of many volume, the tau-leaping algorithm has been modified to handle the communication between volumes. The proposed strategy allows to synchronise volumes by computing a common time increment and using it to select the set of reactions to apply (at each step), inside every membrane, thus resulting in a parallel evolution of the entire system.

A different version of τ-DPP, called Sτ-DPP [29], is also presented. In Sτ-DPP, we allow the communication between non-adjacent membranes, and we associate a measure to membranes and objects, representing the "size" of the volumes where the computation occurs and the volume of each

object, respectively. Both the sizes of membranes and objects are useful to describe any real system where it is important to avoid the infinite accumulation of objects inside a membrane, which is very important in chemical and biological systems, and cannot be achieved by simply bounding the "capacity" of the membranes or by limiting the maximum number objects allowed inside a particular volume.

The frameworks presented in this thesis have been used for the modelling, simulation and analysis of ecological, biological and chemical systems. In particular, we present an application of DPPs for the investigation of *metapopulations*, also called multi-patch systems, which are ecological models used to analyse the behaviour of interacting populations, to the aim of determining how a fragmented habitat influences local and global population persistence.

Then, the framework of τ-DPP has been applied to an extensive study of different biological systems. For instance, we present a discrete mathematical model for the Ras/cAMP/PKA pathway in the yeast *Saccharomyces cerevisiae*, which is involved in the regulation of metabolism and cell cycle progression [31]. We investigate this system under various conditions, and we test how different values of several stochastic reaction constants affect the pathway behaviour.

A model of a genetic oscillator coupled with a quorum sensing intercellular mechanism is also considered as a case study [18]. This intercellular communication mechanism is able to lead the *local* genetic oscillators, within a noisy and nonidentical population, to *global* oscillatory rhythms. In particular, it was shown that individual repressilator systems can self-synchronize, even when their periods are broadly distributed [67]. The multivolume model of this system consists of n volumes, each one corresponding to a cell, enclosed inside an additional volume representing the environment.

Then, the modelling and stochastic simulations of the chemotactic signal transduction pathway in bacteria, are presented [14]. This particular pathway allows bacteria to respond and adapt to environmental changes, by tuning their tumbling and running motions that are due to clockwise and counterclockwise rotations of their flagella. By exploiting the model and the results of simulations obtained with τ-DPP, we investigate the interplay between the stochastic fluctuations of the amount of a particular protein of this pathway and the number of cellular flagella as the core component that stands at the basis of chemotactic motions. The aim of this analysis is to devise the mean time periods during which the cell either performs a running or a tumbling motion, considering both the coordination of flagella and the randomness that is intrinsic in the chemotactic pathway.

A simple biochemical system has been simulated by means of Sτ-DPP. The system describes the transport of molecules from the cytoplasm (modelled as a set of nested membranes) to the nucleus. This task can be accomplished by using simple communication rules or by means of a microtubule

[167], which is a sort of intracellular "highway" that efficiently transport molecules towards the nucleus. Here, by changing the size associated to the membranes representing the microtubule, we highlight the role played by the "space" and its effects on the system's dynamics.

We consider also a different issue, that is related to a possible implementation of τ-DPP by means of chemical systems. To this aim, we introduce the framework of chemical computing, in order to show how to describe "computations" performed with τ-DPP, by means of chemical reactions [164]. In particular, we present the encoding of simple boolean functions, of the Fredkin gate and an instance of Fredkin circuit. The encoding of simple logic gates is useful because by composing them in particular circuits, it is possible to encode any boolean function. Besides this, we also present encodings for register machines instructions, and we give some insights about the construction of a complete register machine with n registers, with the aim to obtain a "parallel device" that can be used, for instance, in the field of computational complexity theory.

Another key issue related to the modelling of biochemical systems regards the calibration of the parameters. For instance, in the τ-DPP applications presented in this thesis, the values of stochastic constants associated to the chemical reaction have been obtained by first assuming plausible relative magnitudes for their values, and then by adjusting them one by one, until a good reproduction of the expected behaviour has been obtained.

In general, many numerical factors are needed for a complete and accurate description of biological systems, like molecular species quantities and reaction rates, which represent an indispensable quantitative information to perform computational investigations of the system behaviour. Unfortunately, the experimental values of these factors are often not available or inaccurate, since carrying out their measurements *in vivo* can be tangling or even impossible [173]. In a few cases, the values of some parameters of a given system can be estimated either from *in vitro* experiments (by fitting the dynamics derived through equations based on mass-action law against the concentration time series that result from these measurements), or by assuming some analogies with similar processes or organisms for which more experimental data are available.

The lack and the inaccuracy of these information bring about the challenging problem of developing suitable techniques to automatically estimate the correct values to all parameters in order to reproduce the expected dynamics in the best possible way.

Optimization methods can be used to tackle the calibration problem of *parameter estimation* of biochemical systems by minimizing a cost function (e.g. a distance measure) which quantitatively defines how good is the system behaviour, using the predicted values, with respect to the experimental dynamics. Several global optimization techniques have already been adopted for parameter estimation of biochemical and biological systems.

In this thesis, we consider the application of two optimisation techniques, *genetic algorithms* and *particle swarm optimizer*, to tackle this problem. To this aim, we provide the definition of a fitness function that is suitable to quantify the quality of a particular set of parameters used during the stochastic simulation of a system. Working in the field of stochastic modelling and simulation, the fitness definition is based on the idea that we have to compare the observed dynamics with the dynamics generated by using a stochastic simulation algorithm, which will run using a particular set of parameter values. Therefore, we have to manage some troublesome properties that are inherent to stochastic simulations like, for instance, the irregular time sampling of the resulting outcome.

In this thesis, in particular, we test and compare the performances of genetic algorithms and particle swarm optimization to the aim of identify the most suitable optimisation technique for the parameter estimation. To this aim, these methods are applied to two simple biochemical schemes, which have been chosen since they are well representative of the dynamics of many other biological systems: a basic catalytic kinetics (the Michaelis-Menten system) and an oscillating behaviour (based on the Belousov-Zhabotinskii reaction) [15]. The oscillating systems has been considered with two different set of parameters which provide two distinct dynamics, characterised by sustained oscillations and a dynamics with damped oscillations, respectively.

Finally, the problem related to the exploration of the parameters space of a biochemical system is described. Usually, this kind of analysis is achieved by means of large numbers of independent simulations where each execution is performed with a particular parametrisation. To efficiently tackle this problem, we present the implementation of a parameter sweep application on a grid framework [140], obtained by distributing large numbers of simulations performed by means of τ-DPP. This grid implementation is used in 4 different parameter sweep applications executed on a model of the chemotactic signal transduction pathway in bacteria composed by 59 chemical reactions, and their performances are analysed and compared.

Overview

This thesis is structured as follows. In the first part (Chapters 1–3) we introduce the prerequisite notions that are necessary for the development of our work. In the second part (Chapter 4–6) we present our approach for the modelling and simulation of biochemical systems, together with some applications, as well as the tool for the analysis of the parameter space of a system.

In Chapter 1, the main stochastic algorithms for the description of the dynamics of biochemical systems are presented. We start by introducing the basic notions and the fundamental hypothesis needed for the develop-

ment and the application of these algorithms. The reference procedure is the *stochastic simulation algorithm* (SSA), whose theoretical basis is exploited in most of the other stochastic algorithms. Afterwards, we present (1) the *next reaction method*, which is faster than SSA since it uses suitable data structure to efficiently handle the computation and re-uses (previously drawn) random numbers; (2) the *tau-leaping* algorithm, an approximate procedure in which reactions are applied in parallel to achieve fast simulations; (3) the *next subvolume method*; (4) the *Binomial tau leap spatial stochastic simulation algorithm* for the simulation of heterogeneous systems composed by a number of sub-volumes.

In Chapter 2, two optimisation techniques are presented: *genetic algorithms* and *particle swarm optimizer*. Genetic algorithms are a population based heuristic which select the individuals for the next generation according to their quality, and evolve them by means of variation operators. Particle swarm optimizer moves a swarm of particles through a n-dimensional space, towards the best position found so far by each particle and by the entire swarm. In the investigation of biochemical systems, both techniques can be applied to tackle the parameter estimation issue, which consists in the calibration of the system's parameters (this particular application is presented in Chapter 6).

In Chapter 3, the framework of *membrane systems*, or P systems, is described. First, the basic notions and definitions of P systems are provided. Then, the variant of *dynamical probabilistic P systems* (DPP) is introduced. DPPs propose a new approach for the application of P systems, which consists in interpreting them as stochastic tools for the description of the dynamics of complex systems. The key feature is that, unlike the basic version of P systems, in DPPs probabilities are associated with the rules (following a method similar to that used in SSA), and these values vary during the evolution of the system according to a prescribed strategy. The "evolution" of a system described by DPPs is therefore governed by a stochastic process. We present an application of DPPs for the modelling and simulation of a multivolume system, called *metapopulations*, which are ecological models that describe the behaviour of interacting populations, to the aim of determining how a fragmented habitat influences local and global population persistence.

In Chapter 4 we present a novel technique for the simulation of complex biochemical systems, which combines the descriptive power of membrane systems with the efficiency of tau-leaping procedure. This method is called τ-DPP, it represents an extension of the DPP variant of membrane systems, since it introduces a strategy for providing quantitative descriptions of a system's dynamics. τ-DPP also extends the applicability of tau-leaping simulation algorithm, as it provides a novel procedure to simulate systems consisting of many volumes, but still relying on the efficiency of the original simulation procedure. A further version of τ-DPP, called Sτ-DPP, is then

presented. It represents an improvement of the previous version since it considers the size of objects (molecules) and compartments (volumes) involved in the system. In this case, the application of a set of rules is enabled only if the compartment where they are applied contains enough free space for the freshly produced (or communicated) objects.

In Chapter 5 different applications of τ-DPP are provided. For each application, the model described by means of τ-DPP framework is presented, along with the results obtained by simulating the system. The first application regards the Ras/cAMP/PKA signalling pathway in the yeast *Saccharomyces cerevisiae*, which is involved in the regulation of metabolism and cell cycle progression. The second application is a model of a genetic oscillator called *repressilator*, coupled with a quorum sensing intercellular mechanism. The third application is related to a model of the *chemotactic signal transduction pathway* in bacteria. Finally, the implementation of boolean gates, circuits and register machine instructions, exploiting the chemical computing theory, is also presented within the framework of τ-DPP.

In Chapter 6 we give a detailed description of the role played by the parameters of a biochemical system, focusing on the stochastic constants associated to the chemical reactions. We present the *parameter estimation* issue, that is, the problem related to the calibration of the system's parameters, and the *parameter sweep application*, a method suitable to explore the space defined by the system's parameters. In particular, we provide the definition of the fitness functions that we use both in parameter estimation to evaluate the quality of a particular set of parameters, and in the parameter sweep to quantify the difference between the "wild type" dynamics and those obtained by using different parametrisations. In particular, we present an implementation of genetic algorithms and particle swarm optimizer to tackle the parameter estimation issue, and we compare their performance by applying them to simple biochemical networks. Then, we propose a grid implementation of the parameter sweep application, obtained by distributing a large number of simulations performed by means of τ-DPP, to the aim of efficiently exploring the parameter space of a biochemical system.

Finally, in Chapter 7 conclusive remarks and a discussion about the presented work are proposed. Insights concerning some possible improvements and future directions for research are also briefly described.

Published works

The work of this thesis is based on the following publications:

D. Besozzi, P. Cazzaniga, D. Pescini, G. Mauri. **A multivolume approach to stochastic modelling with membrane systems.** In: Algorithmic Bioprocesses, (A. Condon, D. Harel, J.N. Kok, A. Salomaa, E. Winfree, eds.), Natural Computing Series, Springer-Verlag, 519-542, 2009.

P. Cazzaniga, G. Mauri, L. Milanesi, E. Mosca, D. Pescini. **A novel variant of tissue P systems for the modelling of biochemical systems.** Proceedings of the 10th International Workshop on Membrane Computing, WMC 2009 (G. Paun, M.J. Perez-Jimenez, A. Riscos-Nunez, G. Rozenberg, A. Salomaa, eds.), to appear in LNCS.

E. Mosca, P. Cazzaniga, D. Pescini, I. Merelli, G. Mauri, L. Milanesi. **Stochastic simulations on a grid framework for parallel sweep applications in biological models.** Accepted for presentation at HiBi09 - High Performance Computational Systems Biology Workshop, 14-16 October 2009 - Trento, Italy (submitted to Briefings in Bionformatics).

D. Besozzi, P. Cazzaniga, M. Dugo, D. Pescini, G. Mauri. **A study on the combined interplay between stochastic fluctuations and the number of flagella in bacterial chemotaxis.** Proceedings of CompMod2009 - 2nd International Workshop on Computational Models for Cell Processes (R.J. Back, I. Petre, E. de Vink, eds.), EPTCS, 6, 47-62, 2009.

P. Cazzaniga, D. Pescini, L. Vanneschi, D. Besozzi, G. Mauri. **A comparison of genetic algorithms and particle swarm optimization for parameter estimation in stochastic biochemical systems.** Proceedings of EvoBio 2009 (C. Pizzuti, M.D. Ritchie, M. Giacobini, eds.), LNCS 5483, 116-127, 2009.

D. Pescini, P. Cazzaniga, C. Ferretti, G. Mauri. **First steps toward a wet implementation for tau-DPP.** Proceedings of the 9th International Workshop on Membrane Computing, WMC 2008 (D. W. Corne, P. Frisco, G. Paun, G. Rozenberg, A. Salomaa, eds.), LNCS 5391, 355-373, 2009.

P. Cazzaniga, D. Pescini, D. Besozzi, G. Mauri, S. Colombo, E. Martegani. **Modeling and stochastic simulation of the Ras/cAMP/PKA pathway in the yeast *Saccharomyces cerevisiae* evidences a key regulatory function for intracellular guanine nucleotides pools.** Journal of Biotechnology, 133, 3, 377-385, 2008.

D. Besozzi, P. Cazzaniga, D. Pescini, G. Mauri. **Modelling metapopulations with stochastic membrane systems.** BioSystems, 91, 3, 499-514, 2008.

D. Besozzi, P. Cazzaniga, D. Pescini, G. Mauri. **Seasonal variance in P system models for metapopulations.** Progress in Natural Science, 17, 4, 392-400, 2007.

P. Cazzaniga, D. Pescini, D. Besozzi, G. Mauri. **Tau leaping stochastic simulation method in P systems.** Proceedings of the 7th International Workshop on Membrane Computing, WMC 2006 (H.J. Hoogeboom, G. Paun, G. Rozenberg, A. Salomaa, eds.), LNCS 4361, 298-313, 2006.

Chapter 1

Stochastic algorithms for the simulation of biochemical systems

In this chapter, some of the most used and well known stochastic simulation techniques will be presented. These techniques can be used to describe the dynamics of biochemical systems with fixed conditions (i.e. pressure, temperature, etc.), in which the set of molecular species involved, and the possible interactions between species (chemical reactions) are known. In particular two exact procedures, the *stochastic simulation algorithm* [74] and the *next reaction method* [72] will be presented. These two algorithms can be applied to homogeneous systems, that is, systems in which the molecules are considered well mixed, and they provide a sequential description of the system's dynamics, by executing *one* reaction during each simulation step. The former is Gillespie's seminal work, which represents the basis of most of the stochastic algorithms for the simulation of chemical systems; the latter has been introduced by Gibson and Bruck as a faster version of the stochastic simulation algorithm. The exactness of these two methods refers to the simulated evolution of the analysed system, which corresponds to an "exact" numerical realisation of the actual dynamics of the system.

An approximate simulation technique called *tau-leaping*, firstly presented in [77], will be then described. This method has been developed in order to speed up stochastic simulations, in fact, *many* reactions are executed during each iteration. On the other hand, with this method there is a loss in the accuracy of the simulated dynamics with respect to the stochastic simulation algorithm. In the following, we will refer in particular to the tau-leaping version presented in [26].

Finally, two stochastic algorithms used to simulate heterogeneous systems will be illustrated. These procedures are called *next subvolume method* [57] and *binomial tau-leap spatial stochastic simulation algorithm* [117]. The-

se procedures consider both reactive and diffusive processes, which occur inside the adjacent *subvolumes* constituting the analysed system. In particular, the former method is based on the stochastic simulation algorithm, and simulates one event for each iteration step; the latter is based on the tau-leaping, hence many reaction and diffusive events are simulated during each step.

1.1 Gillespie's stochastic simulation algorithm

In this section, the *stochastic simulation algorithm* (SSA) will be presented. This simulation technique has been introduced by Gillespie [73, 74] in order to reproduce the exact behaviour of chemical systems. SSA represents the reference point for the development of new procedures, and it is one of the most used algorithms for the description of the dynamics of chemical and biological systems. As a matter of fact, SSA has been implemented within many software tools [188, 41, 50, 33].

SSA can be applied to chemical systems defined within a single volume Ω, which is assumed to be well stirred and at constant temperature. This is the fundamental assumption that leads to the possibility of describing the system behaviour, without considering the position and velocity of every single molecule occurring inside the system. Inside Ω, a set of chemical species $\{S_1, \ldots, S_N\}$, whose interactions are governed by a set of chemical reactions $\{R_1, \ldots, R_M\}$, is considered. A stochastic constant c_j $(j = 1, \ldots, M)$, which depends only on the chemical and physical properties of the molecules and on the temperature of the system, is associated to each reaction R_j. The number of molecules of species S_i $(i = 1, \ldots, N)$ at a given time t is denoted by $X_i(t)$, and the state of the system at time t is defined as the vector $\mathbf{X}(t) \equiv (X_1(t), \ldots, X_N(t))$.

It is clear that the changes in the molecular numbers of the species involved in the systems are a consequence of the application of the chemical reactions. A chemical reaction R_j has the general form $\alpha_1 S_1 + \cdots + \alpha_N S_N \xrightarrow{c} \beta_1 S_1 + \cdots + \beta_N S_N$ where the stoichiometric coefficients α_i denote the number of molecules S_i consumed by reaction R_j (i.e. *reagents*), while the stoichiometric coefficients β_i denote the number of molecules S_i produced by R_j (i.e. *products*).

The effects of the application of a chemical reaction are summarized by means of the *state-change vector* $\mathbf{v}_j \equiv (v_{1j}, \ldots, v_{Nj})$ $(j = 1, \ldots, M)$. The element v_{ij} of $\mathbf{v}_j$ represents the multiplicity change of the species S_i due to reaction R_j.

Given the system state of the system $\mathbf{X}(t) = \mathbf{x}$, the propensity function $a_j(\mathbf{x})$ of the reaction R_j is defined as the probability that *one* application of such reaction will occur inside Ω in the infinitesimal time interval $[t, t + dt)$. The propensity functions represent the stochastic "rates" of the reactions

involved in a system, and they are used to select the time increment and the reaction to execute in order to describe the system's dynamics, as it will be explained in the following.

The definition of the propensity function a_j is derived by considering the existence of a constant c_j such that $c_j dt$ gives the probability that a particular molecule (in the case of unimolecular reactions) or a randomly chosen combination of molecules (in the case of reactions of higher order) will react in the next infinitesimal time interval.

Starting from this consideration, in the case of a first order reaction $R_j : S_i \xrightarrow{c_j} products$, having X_i molecules of species S_i, the probability that one of them will be transformed by means of the R_j, in the infinitesimal time interval dt, is given by the product of X_i by the stochastic constant c_j, that is, $a_j(\mathbf{x}) = X_i c_j dt$.

If the considered reaction R_j is bimolecular, having the form $R_j : S_i + S_l \xrightarrow{c_j} products$, the probability that one pair of molecules of the species S_i, S_l will react in the next time interval is given by $a_j(\mathbf{x}) = X_i X_l c_j dt$, because each possible combination of molecules of the species which undergo to R_j has to be considered.

In the case of a reaction $R_j : S_i + S_i \xrightarrow{c_j} products$, that is, a bimolecular reaction which involves two molecules of the same species S_i, then the total number of possible pairs is $\frac{1}{2}X_i(X_i - 1)$ and the propensity function is given by $a_j(\mathbf{x}) = \frac{1}{2}X_i(X_i - 1) \cdot c_j dt$.

The aim of the SSA is to evaluate the value of $\mathbf{X}(t) = \mathbf{x}$, given an initial state $\mathbf{X}(t_0) = \mathbf{x}_0$ at time t_0. This could be done through the estimation of $P(\mathbf{x}, t|\mathbf{x}_0, t_0)$, which represents the probability that the system will be in the state $\mathbf{X}(t) = \mathbf{x}$, starting from $\mathbf{X}(t_0) = \mathbf{x}_0$. From this probability value, a time evolution equation can be derived, exploiting the notion of propensity function and of the state-change vectors:

$$\frac{\partial P(\mathbf{x}, t|\mathbf{x}_0, t_0)}{\partial t} = \sum_{j=1}^{M} \left[a_j(\mathbf{x} - \mathbf{v}_j) P(\mathbf{x} - \mathbf{v}_j, t|\mathbf{x}_0, t_0) - a_j(\mathbf{x}) P(\mathbf{x}, t|\mathbf{x}_0, t_0) \right]. \tag{1.1}$$

This equation, which completely determines the function $P(\mathbf{x}, t|\mathbf{x}_0, t_0)$, is called the Chemical Master Equation (CME). The problem is that the CME is composed by a set of coupled ordinary differential equations (ODEs), with one equation for each possible combination of reactant molecules occurring in the system. Therefore, the CME can be analytically solved only for simple cases, while for others, the computational burden makes the numerical solution impossible to compute.

The impracticability of finding a solution for the CME, which consists in computing the probability density function $\mathbf{X}(t)$, has led to the development

of a procedure that allows to generate a numerical realization of the system's evolution, namely, a simulated trajectory of $\mathbf{X}(t)$ in time. Note that, this is different from determining the solution of the CME because, instead of computing the probability density function, a random sample of $\mathbf{X}(t)$ will be found.

In fact, the trajectory $\mathbf{X}(t)$ of the system can be generated starting from another probability function, $p(\tau, j|\mathbf{x}, t)$, which represents the probability that, given the state $\mathbf{X}(t) = \mathbf{x}$, the next reaction occurring in the system will be R_j, and the time interval for its execution will be $[t + \tau, t + \tau + d\tau)$.

Given the current system state $\mathbf{x}$, the function p is the joint probability density function of the two random variables τ and j which denote the occurrence time to the next reaction and the index of the next reaction that will be executed, respectively. Applying the laws of probability to the propensity function defined above, an exact formula for the function p can be derived as:

$$p(\tau, j|\mathbf{x}, t) = a_j(\mathbf{x})\ e^{(-a_0(\mathbf{x})\tau)} \tag{1.2}$$

where a_0 is the sum of all the propensity functions of the reactions R_m ($m = 1, \ldots, M$), defined as $a_0(\mathbf{x}) = \sum_{m=1}^{M} a_m(\mathbf{x})$. Equation 1.2 states that τ is an exponential random variable with mean and standard deviation equal to $1/a_0(\mathbf{x})$, and j is a statistically independent integer random variable with point probabilities $a_j(\mathbf{x})/a_0(\mathbf{x})$. Note that, this is the starting point for a stochastic simulation, and using an exact Monte Carlo procedure for generating samples of τ and j according to their distributions, an exact trajectory of $\mathbf{x}$ can be generated [75].

The simplest procedure with the aim to generate a numerical realisation of the system is the so-called *direct method*, in which two random numbers (r_1 and r_2) are tossed from a uniform distribution in the unit interval $[0, 1]$ and the values of τ and j are computed as follows:

$$\tau = \frac{1}{a_0(\mathbf{x})}\ ln\left(\frac{1}{r_1}\right), \tag{1.3}$$

$$\sum_{m=1}^{j-1} a_m(\mathbf{x}) < r_2 a_0(\mathbf{x}) \leq \sum_{m=1}^{j} a_m(\mathbf{x}). \tag{1.4}$$

Equation 1.3 generates a random number τ according to the probability density function $p_1(\tau) = a_0(\mathbf{x})\ exp(-a_0(\mathbf{x})\tau)$, while Equation 1.4 generates a random integer j according to the probability density function $p_2(j) = a_j(\mathbf{x})/a_0(\mathbf{x})$ (considering that $p_1(\tau) \cdot p_2(j) = p(\tau, j)$), as explained in [74].

Following this method for the generation of the values of τ and j, the functioning of the SSA - describing an exact numerical realization of the process $\mathbf{X}(t)$ - can be formalised as follows [73, 74]:

1. Initialize the system time, $t = t_0$, and set the initial amount of the molecular species, $\mathbf{x} = \mathbf{x}_0$. Load the reactions involved in the system along with their stochastic constants;

2. Evaluate the propensity functions $a_j(\mathbf{x})$ of the reactions, according to the system state $\mathbf{x}$, and their sum $a_0(\mathbf{x})$;

3. Generate the values for τ and j using Equations 1.3 and 1.4;

4. Execute reaction j by updating the system state as $\mathbf{x} = \mathbf{x} + \mathbf{v}_j$, and update the system time as $t = t + \tau$;

5. If the termination criterion is satisfied, then end the simulation. Otherwise go back to *Step 2*.

During each iteration, this procedure computes the values of the propensity functions of the reactions, according to the current system state. Then, the values of τ and j are computed as described by Equations 1.3 and 1.4. The final step consists in the system update according to the state-change vector of the selected rule. The termination criterion usually used in this kind of procedure regards the number of executed iterations or the simulated time.

The method is called "direct", because it generates directly the values of τ and j.

There exists an alternative procedure to implement the SSA, which is equivalent to the direct method, that can be formalised by the following iterative steps [73, 74]:

1. Initialize the system time, $t = t_0$, and set the initial amount of the molecular species, $\mathbf{x} = \mathbf{x}_0$. Load the reactions involved in the system along with their stochastic constants;

2. Evaluate the propensity functions $a_j(\mathbf{x})$ of the reactions, according to the system state $\mathbf{x}$;

3. Generate a candidate τ_j, for each reaction R_j, according to an exponential distribution based on a_j, such that $\tau_j = 1/a_j(\mathbf{x})\ ln(1/r_j)$;

4. Select j with the smallest candidate τ_j, and let $\tau = \tau_j$;

5. Execute reaction j by updating the system state as $\mathbf{x} = \mathbf{x} + \mathbf{v}_j$, and update the system time as $t = t + \tau$;

6. If the termination criterion is satisfied, then end the simulation. Otherwise go back to *Step 2*.

This procedure is called *first reaction method*, the main difference from the direct method is that here M different random numbers, where M is the number of reactions, are needed at each iteration, in order to compute the value of τ.

In general, SSA has many advantages with respect to other (standard) simulation algorithms, such as: (1) the procedure, that is logically equivalent to the CME, is exact and takes full account of the stochastic fluctuations of the system; (2) the length of the step τ is exact and not a finite approximation of some infinitesimal dt, as the time step increment used, for instance, by ODEs solvers; (3) the procedure is very easy to codify, and it does not depend on the set of reactions involved in the system. The amount memory required for a simulation is typically small, i.e., it is proportional to the number of molecular species and chemical reactions; (4) while the CME tries to solve the system simultaneously for the probability of all possible trajectories, SSA generates a single trajectory, therefore it can easier describe the dynamics of a system; (5) by using a number of independent runs of SSA, it is easy to calculate means, variances, correlations, etc. of the species involved in the system. Note that, these ensemble averages cannot be readily computed using the CME.

On the other hand, SSA also presents weaknesses: (1) the computational time required to run a single simulation, which is proportional to the number of reactions M, is usually high because of the sequential execution of a single reaction at each iterative step; (2) the computational time is also proportional to the number of molecules involved in a system, hence, there is a limitation either in the molecular amounts which can be considered, or in the total simulation time which can be considered during each execution; (3) the ensemble averages that can be computed from a number of independent simulations is typically very time consuming. Hence, the statistical accuracy is directly dependent on the time required to execute the simulations.

SSA has been used for the simulation of many biological and chemical systems. The earliest application of the SSA to a real biological system demonstrating that stochasticity can play a critically important role has been presented in [123], where a model of the mechanism controlling gene transcript and translation has been analysed. Another SSA application regards the study of the effect of fluctuations in gene expression rates and other molecular–level fluctuations on lysis or lysogeny pathway selection statistics by phage λ–infected *Escherichia coli* cells [6].

1.2 The next reaction method

In this section, a procedure developed by Gibson and Bruck [72] for the optimisation of the SSA, will be presented. This algorithm, called *next reaction method* (NRM), overcomes one of the main limitations of SSA,

that is, its applicability to systems with a large number of molecular species and chemical reactions. In fact, in such cases the computational time of SSA usually becomes prohibitively long.

The NRM is based on the method introduced by Gillespie and it is exact as the SSA, though it is more efficient than SSA. NRM exploits the first reaction method, however, instead of using a random number for each reaction (at each iteration), it assigns a single random number per reaction event. Moreover, the efficiency of the method relies both on the data structures used to store the propensity function values and the drawn random numbers, and on the way they are updated, when needed. Note that, the improvements due to the use of these data structures can be also extended to the Gillespie's direct method.

The aim of NRM is to avoid the execution of three particular operations repeated at each iteration of the first reaction method: the update of the random numbers associated to the reaction events, the computation of the candidate τ_i related to each reaction, and the identification of the smallest value, among τ_i's, which represents the actual τ.

The main idea is to store the values of τ_i along with the values of the propensity functions a_i of the reactions, and to recalculate the values of a_i (and of the corresponding τ_i) only if they change. To realise this optimised update operation, a *dependency graph* is used. This data structure indicates which is the relation between reactions and propensity functions, namely, it indicates which propensity functions are affected by the execution of a reaction.

The values of τ_i which are not affected by the reaction executed during an iteration are not modified and re-used at the next iteration. In general, the random numbers used during Monte Carlo simulations cannot be re-used; nevertheless, in this particular case it is legitimate, as proved in [72]. Hence, all the τ_i, except for τ_j (used for the selected reaction), will be re-used. Note that, so doing, only a few values of both a_i and τ_i will be updated at each iteration, therefore it is suitable to use an efficient data structure to store the values and to update them (when needed). This structure is called *indexed priority queue*.

Hereafter, the definitions of dependency graph and indexed priority queue will be provided.

In order to define the dependency graph of a chemical system, the identification of *reactants* and *products*, of a given reaction, is needed. Given the reaction R_i, these two sets of molecular species will be called $reactants(R_i)$ and $products(R_i)$, respectively. For instance, let us consider the reaction $R_i : A + B \rightarrow C$, then $reactants(R_i) = \{A, B\}$ and $products(R_i) = \{C\}$. Moreover, the set of molecular species which affect the value of a_i is called $dependsOn(a_i)$. Note that, usually $dependsOn(a_i) = reactants(R_i)$, but there are cases in which some additional information can be added. Finally, the set of molecular species whose quantities are changed after a rule

Table 1.1: A simple chemical system.

Reaction	$dependsOn(a_r)$	$affects(r)$
$A + B \rightarrow C$	A, B	A, B, C
$C \rightarrow A + B$	C	A, B, C
$B + C \rightarrow D$	B, C	B, C, D
$D + E \rightarrow F + E$	D, E	D, F
$A + E \rightarrow D$	A, E	A, D, E

execution is called $affects(R_i)$. In general, $affects(R_i) = reactants(R_i) \cup products(R_i)$, however, there exist particular kinds of reactions, like catalytic processes, where the set of affected molecular species is defined differently. For instance, given $R_i : D + E \rightarrow F + E$, then $affects(R_i) = \{D, F\}$ (while $reactants(R_i) \cup products(R_i) = \{D, E, F\}$).

Starting from the sets listed above, the *dependency graph* associated to a biochemical system can be defined as follows. Let $\mathbb{G}(V, E)$ be a graph where $V = \{v_1, \ldots, v_M\}$ is a set of vertices whose elements corresponds to the chemical reactions of the given system (i.e. $V \equiv R$), and there exists an edge in E, connecting v_i to v_j, if and only if $affects(v_i) \cap dependsOn(a_{v_j}) \neq \emptyset$. Moreover, the self-edge of each vertex have to be added to E.

The dependency graph is a suitable data structure used to indicate the propensity functions which have to be recalculated after a reaction execution. For instance, given the set of reactions listed in Table 1.1, the corresponding dependency graph is shown in Figure 1.1.

In general, the number of edges of a dependency graph is small, meaning that only a limited number of propensity functions and τ's need to be recalculated, at each iteration. A data structure suitable to efficiently handle the operations on the stored values is the *indexed priority queue*. In particular, regarding the a_i values, the only operations needed are $read$ and $update$, hence, they can be stored in an array. On the other hand, the τ values require a $find\_min$ operation, to find the smallest value currently stored, and an *update* operation. Note that, $find\_min$ represents one of the standard operation involved in the priority queues [42] (typically implemented as heap), and also the *update* operation can be implemented using standard operations of priority queues like *add_element* and *delete_element*. However, in this context, a more efficient *update* operation can be implemented by exploiting a particular indexing scheme.

The *indexed priority queue* is defined using two data structures. The first one is a *tree structure* whose nodes consist in ordered pairs (i, τ_i), where i is the reaction index and τ_i is the candidate τ associated to the reaction R_i. Note that, the nodes of the tree have the property that each parent has a τ value smaller than the values stored in its children nodes. The second

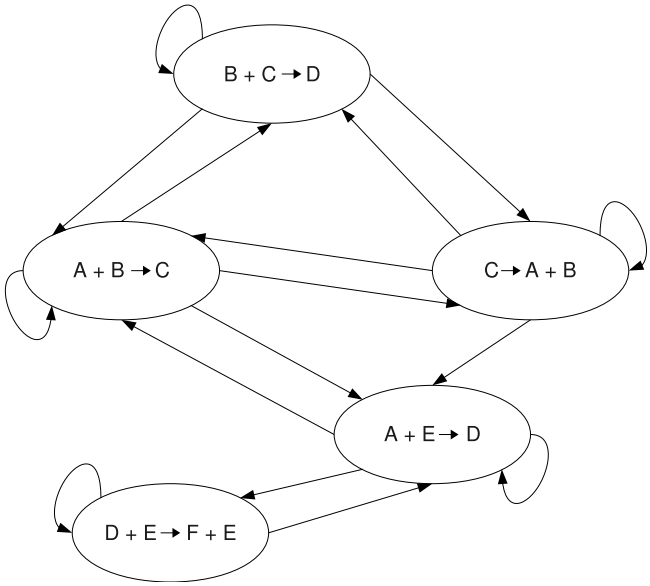


Figure 1.1: Dependency graph corresponding to the set of reactions listed in Table 1.1.

data structure is an *index structure* $\mathbb{P}$ where each element i is a pointer to the node of the tree where the pair (i, τ_i) is stored. An example of indexed priority queue is reported in Figure 1.2.

The advantages of the indexed priority queue are the following: (1) the time needed to retrieve the smallest τ value is constant, because this value is always in the root of the tree; (2) the ordering of the nodes of the tree is only vertical and not horizontal; (3) the number of nodes is equal to the number of reactions; (4) thanks to the indexing scheme, any reaction is identified in constant time.

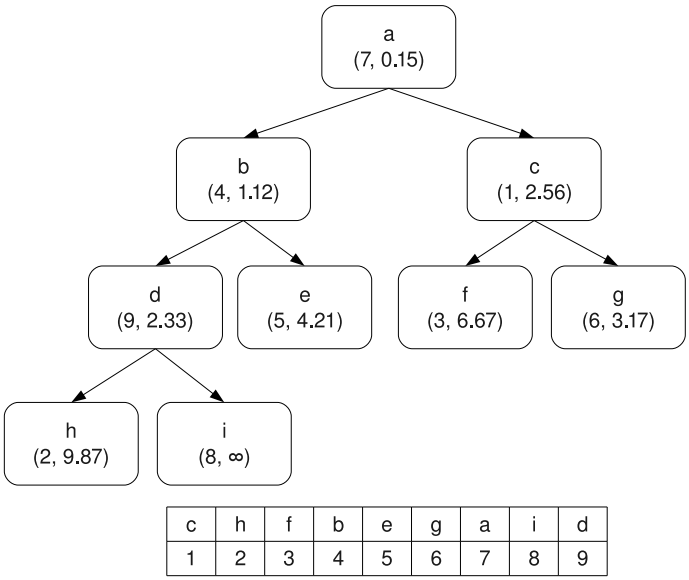
Using the data structures previously explained, the NRM can be formalised as follows:

1. System initialization. Set the initial amounts for the molecular species, set the time $t = t_0$, and generate the dependency graph $\mathbb{G}$. Compute the propensity function a_i, for all reactions i. For each reaction i, generate a candidate time τ_i, according to an exponential distribution with parameter a_i. Store the τ_i values in an indexed priority queue $\mathbb{P}$;

2. Identify the reaction j with the least value τ_j;

3. Set the time to next reaction $\tau = \tau_j$;

4. Update the state of the system according to the selected reaction, modifying the amounts of the molecular species involved in the reaction.



c	h	f	b	e	g	a	i	d
1	2	3	4	5	6	7	8	9

Figure 1.2: The tree structure containing the pairs (i, τ_i) related to the reactions (above), and the indexing scheme (below).

5. For each edge (j, l) in the dependency graph $\mathbb{G}$ do:

   - update the value of a_l;
   - if $l \neq j$, then set $\tau_l = (a_{l,old}/a_{l,new})(\tau_j - t) + t$;
   - if $l = j$, then generate a random number rnd_j, according to an exponential distribution with parameter a_j and set $\tau_j = rnd_j + t$;
   - replace the old value in $\mathbb{P}$, with the new value of τ;

6. If the termination criterion is satisfied, then end the simulation. Otherwise go back to *Step 2*.

The NRM algorithm needs a computational time proportional to $logM$, where M is the number of reactions occurring in the system. As in the case of SSA, the usual termination criteria are related to the number of reactions executed and to the total time simulated. Note that, a more efficient computation is achieved if the dependency graph is sparse, namely, the number of edges is small with respect to the number of nodes. In Table 1.2, the comparison of the number of operations executed by SSA and NRM, during the simulation of a test case model composed by 10 reactions, is reported to show the gain of efficiency given by NRM (data taken from [72]).

Table 1.2: Comparison of SSA and NRM in terms of number of operation executed (all numbers are in millions).

Operation	SSA	NRM
a_i computation	2700	210
$\times$ and $\div$ ops	0	340
$+$, $-$ and comparisons	2900	1100
exp rnd numbers	35	35
uni rnd numbers	35	0

An example of application of the NRM has been presented in [71], where a Lambda model related to gene transcription and translation, protein–protein interactions and feedback via protein–DNA binding has been simulated and analysed.

1.3 The tau-leaping algorithm

The tau-leaping algorithm was first introduced in [77] to the aim of speeding up stochastic simulations of biochemical systems. Instead of simulating the dynamics of the system by tracing every single reaction event occurring inside the volume Ω, with tau-leaping a time increment τ is computed and a certain number of reactions are selected and executed in parallel. So doing, faster simulations can be performed, though the obtained dynamics of the chemical system is not exact, as in SSA, but it is approximated.

Several different versions of the tau-leaping algorithm have been proposed, aimed at improving the procedure to compute the τ value and to select the reactions to be applied in the current step, avoiding the possibility to obtain negative population of chemical species (we refer to [78, 35, 25] for more details). Despite the improvements in the simulation method achieved by these techniques, they all present a problem related to the description of the system dynamics: though an error control parameter is used, they do not allow to uniformly bound the changes of the species quantities during the τ selection procedure, therefore resulting in a poor approximation of the system dynamics. Moreover, in order to compute the time increment at each step, they require the evaluation of a quadratic number of auxiliary quantities (relative to the number of chemical reactions).

These problems have been worked out in the tau-leaping version presented in [26], which will be considered hereafter. As already said, the aim of this procedure is to fire more than one reaction for each time increment $[t, t + \tau)$ in order to speed up the simulations. The determination of the exact probability distribution of the reactions applications, within a generic

step of length τ, is a hard task to solve. Therefore, in order to obtain an efficient numerical realisation of the system's trajectory, the exact dynamics of the system has to be approximated.

Given the state $\mathbf{x}$ of the system, let $K_j(\tau, \mathbf{x}, t)$ be the exact number of times that a reaction R_j will be fired in the time interval $[t, t+\tau)$, so that $\mathbf{K}(\tau, \mathbf{x}, t)$ is the exact probability distribution vector (having $K_j(\tau, \mathbf{x}, t)$ as elements). For arbitrary values of τ, it is difficult to compute the values of $K_j(\tau, \mathbf{x}, t)$. On the contrary, if τ is small enough that the change in the system's state during $[t, t+\tau)$ is so slight that no propensity function will suffer an appreciable change in its value (this is called the *leap condition*), then we can evaluate a good approximation of $K_j(\tau, \mathbf{x}, t)$ by using the Poisson random variable with mean and variance $a_j(\mathbf{x})\tau$.

So, after the computation of a τ value that satisfies the leap condition, it is possible to update the state of the system at time $t+\tau$ according to:

$$\mathbf{X}(t+\tau) = \mathbf{x} \; + \; \sum_{j=1}^{M} \mathbf{v}_j P_j(a_j(\mathbf{x}), \tau) \tag{1.5}$$

where $P_j(a_j(\mathbf{x}), \tau)$, with $j = 1, \ldots, M$, denotes an independent sample of the Poisson random variable with mean and variance $a_j(\mathbf{x})\tau$.

Each iterative step of the tau-leaping procedure is based on four main stages:

1. Generate the maximum changes of each species that satisfy the leap condition.

2. Compute the mean and variance of the changes of the propensity functions.

3. Calculate the τ value.

4. Sample the reactions numbers to apply.

Hereafter, we describe in detail the motivations and the aims of each of the four stages.

1. Satisfying the leap condition. The procedure for the selection of τ is accomplished in order to bound the relative changes in the molecular amounts, in such a way that the relative changes in the propensity functions will be all bounded - during the τ interval - by a small value ε ($0 \leq \varepsilon \leq 1$).

Let $\Delta_\tau X_i$ be the change in the amount X_i of species S_i, during the time interval $[t, t+\tau)$. Given the state $\mathbf{x}$ and its projections $x_i = X_i(t)$, the leap condition can be formalised as:

$$|\Delta_\tau X_i| \; \leq \; max\{\varepsilon_i x_i, 1\} \;\; \text{with } i = 1, \ldots, N, \tag{1.6}$$

where the values $\varepsilon_i = \varepsilon_i(\varepsilon, x_i)$ are chosen so that the relative changes in the propensity functions will be all bounded, at least, by ε.

To do that, we first determine the highest order of reaction in which each species S_i ($i = 1, \ldots, N$) appears as a reactant (denoted by $HOR(i)$). Then compute:

$$\varepsilon_i = \frac{\varepsilon}{g_i} \tag{1.7}$$

where $g_i = g_i(x_i)$ is defined as follows:

1. if $HOR(i) = 1$ then $g_i = 1$

2. if $HOR(i) = 2$ then
$$g_i = \begin{cases} 2 & \text{if } R_i : S_i + S_k \to \ldots \quad \text{with } i \neq k \\ \left(2 + \dfrac{1}{x_i - 1}\right) & \text{if } R_i : S_i + S_i \to \ldots \end{cases}$$

3. if $HOR(i) = 3$ then
$$g_i = \begin{cases} 3 & \text{if } R_i : S_i + S_k + S_l \to \ldots \quad \text{with } i \neq k \neq l \\ \dfrac{3}{2}\left(2 + \dfrac{1}{x_i - 1}\right) & \text{if } R_i : S_i + S_i + S_k \to \ldots \quad \text{with } i \neq k \\ \left(3 + \dfrac{1}{x_i - 1} + \dfrac{2}{x_i - 2}\right) & \text{if } R_i : S_i + S_i + S_i \to \ldots \end{cases}$$

The g_i values corresponding to reactions having $HOR > 3$ can be easily computed by taking into account the combinatoric of the reactant species involved in the reactions.

2. Compute mean and variance. To compute the largest value of τ that satisfies the leap condition (Equation 1.6), we need to evaluate two auxiliary quantities: the mean and the variance of the expected changes in the propensity functions.

Referring to the basic tau-leaping formula (Equation 1.5), it is possible to consider the quantity defined in Equation 1.6 to be:

$$\Delta_\tau X_i = \sum_{j \in J_{ncr}} v_{ij} P_j(a_j(\mathbf{x}), \tau) \quad \text{with } i = 1, \ldots, N, \tag{1.8}$$

where J_{ncr} denotes the set of noncritical reactions.

A *critical reaction* is a reaction with positive propensity function such that a small number of firings is currently left before exhausting one of its reactants. All the other reactions are named, instead, *noncritical reactions*. It is clear that the set of reactions of the system is the direct sum of the critical J_{cr} and noncritical J_{ncr} reactions sets. The motivations of this partition and the choice of $j \in J_{ncr}$ in Equation 1.8, can be found in [26].

As previously said, the Poisson random variables $P_j(a_j(\mathbf{x}), \tau)$ on the right-hand side of Equation 1.8 are statistically independent and have mean

and variance $a_j(\mathbf{x})\tau$. Hence, the mean and variance of their linear combination can be computed as follows:

$$\langle \Delta_\tau X_i \rangle = \sum_{j \in J_{ncr}} v_{ij}[a_j(\mathbf{x})\tau], \quad var\{\Delta_\tau X_i\} = \sum_{j \in J_{ncr}} v_{ij}^2[a_j(\mathbf{x})\tau] \quad (1.9)$$

for $i = 1, \ldots, N$. Hence, following the same reasoning that was used in the τ selection introduced in [78], it is possible to consider the bound given in Equation 1.5 as substantially satisfied if it is simultaneously satisfied by the absolute mean and the standard deviation of $\Delta_\tau X_i$:

$$|\Delta_\tau X_i| \leq max\{\varepsilon_i x_i, 1\}, \quad \sqrt{var\{\Delta_\tau X_i\}} \leq max\{\varepsilon_i x_i, 1\}, \quad (1.10)$$

for $i = 1, \ldots, N$.
Now, substituting formulas 1.9 into conditions 1.10, the following bounds on τ can be derived:

$$\tau \leq \frac{max\{\varepsilon_i x_i, 1\}}{|\sum_{j \in J_{ncr}} v_{ij} a_j(\mathbf{x})|}, \quad \tau \leq \frac{max\{\varepsilon_i x_i, 1\}^2}{\sum_{j \in J_{ncr}} v_{ij}^2 a_j(\mathbf{x})} \quad (1.11)$$

for $i = 1, \ldots, N$.
Finally, it is possible to compute, as described in [78], the two quantities:

$$\mu_i(\mathbf{x}) = \sum_{j \in J_{ncr}} v_{ij} a_j(\mathbf{x}), \quad \forall i \in I, \quad (1.12)$$

$$\sigma_i^2(\mathbf{x}) = \sum_{j \in J_{ncr}} v_{ij}^2 a_j(\mathbf{x}), \quad \forall i \in I, \quad (1.13)$$

where the restriction on the set of noncritical reactions J_{ncr} is still present, due to the conditions of the modified non-negative Poisson τ-leaping [26].
3. Compute the τ value. The leap length is obtained substituting Equations 1.12 and 1.13 into Equation 1.11:

$$\tau = \min_{i \in I} \left\{ \frac{max\{\varepsilon x_i / g_i, 1\}}{|\mu_i(\mathbf{x})|}, \frac{max\{\varepsilon x_i / g_i, 1\}^2}{\sigma_i^2(\mathbf{x})} \right\}, \quad (1.14)$$

where g_i is given by Equation 1.7.

It is also possible to estimate the *mean* $\mu_j(\mathbf{x})\tau$, and the *standard deviation* $\sqrt{\sigma_j^2(\mathbf{x})\tau}$ of the expected change in the propensity function $a_j(\mathbf{x})$ in the time increment τ. Formula 1.14 requires that these quantities would be bounded by $\varepsilon a_j(\mathbf{x})$ for $j = 1, \ldots, M$, thus satisfying the leap condition.
4. Sampling the reactions numbers. The last stage consists in sampling the numbers of firings of each reaction R_j according to the Poissonian distributions $P(a_j(\mathbf{x}), \tau)$ with mean and variance $a_j(\mathbf{x})\ \tau$.

Following the four stages explained above, the tau-leaping algorithm can be formalised as follows (in the description, the algorithm execution naturally proceeds according to the order of instructions, when not otherwise specified by means of "go to" commands).

1. Initialize the system time, $t = t_0$, and set the initial amount of the molecular species, $\mathbf{x} = \mathbf{x}_0$. Load the reactions involved in the system along with their stochastic constants;
2. Locate the set of all critical reactions;
3. Compute the quantities μ_i and ${\sigma_i}^2$;
4. Select the value of τ' as indicated in Equation 1.14;
5. If $\tau' < n/a_0$ (where $a_0 = \sum_{j=1}^{M} a_j(\mathbf{x})$ and the factor n is usually set to 10), then go to *Step 6*. Otherwise go to *Step 7*;
6. Execute an SSA step, exploiting the direct method, as described in Section 1.1; then go to *Step 12*;
7. Compute the sum of the propensity functions of all critical reactions, denoted by $a_0^c(\mathbf{x})$;
8. Generate $\tau'' = 1/a_0^c(\mathbf{x}) \cdot 1/rnd$, where rnd is a random number sampled from the uniform distribution in the unit interval $[0, 1]$;
9. If $\tau' < \tau''$ then go to *Step 10*. Otherwise go to *Step 11*;
10. Set $\tau = \tau'$ and do:
    - for all critical reactions R_j set the number of firings $k_j = 0$;
    - for all noncritical reactions R_j generate k_j as a sample of the Poisson random variable $P(a_j(\mathbf{x}), \tau)$ with mean $a_j(\mathbf{x})\tau$;
    - go to *Step 12*;
11. Set $\tau = \tau''$, and do:
    - select one critical reaction R_j to be fired during this step and set $k_j = 1$; for all other critical reactions R_j set $k_j = 0$;
    - for all noncritical reactions R_j generate k_j as a sample of the Poisson random variable $P(a_j(\mathbf{x}), \tau)$ with mean $a_j(\mathbf{x})\tau$;
12. Update the state of the system: $\mathbf{X}(t + \tau) = \mathbf{X}(t) + \sum_{j=1}^{M} k_j \cdot \mathbf{v_j}$;
13. If the termination criterion is satisfied, then end the simulation. Otherwise go back to *Step 2*.

After the system initialisation, the iterative part of the algorithm starts. During *Step 2*, the set of reactions is divided into two subsets containing critical and non critical reactions. In *Step 3* the procedure calculates the auxiliary quantities needed for the evaluation of the first candidate for the

length of the leap τ' (performed in *Step 4*), which is the largest value that satisfies the leap condition.

During *Step 5*, the algorithm checks if the execution of a tau-leaping step is allowed. In fact, if τ' is less than a multiple of $1/a_0$, then an SSA step is executed because, given the actual state of the system, it will be more accurate and efficient than a tau-leaping step.

In case of the execution of a tau-leaping step, the algorithm proceeds to *Step 7* for the computation of the sum of the propensity functions of the critical reactions, and then to *Step 8* to evaluate the second candidate for the length of the leap τ''.

During *Step 9* the procedure compares the values of τ' and τ''. If τ' is the smallest one, than only non critical reactions will be selected for this iteration (*Step 10*). Otherwise, besides non critical reactions, also one critical reaction will be randomly extracted during the current iteration (*Step 11*).

Finally, in *Step 12* the system state is updated and in *Step 13* the termination criterion is checked, and if the condition holds, then the execution is terminated. Usually, the termination criteria regards the number of iteration executed or the total time simulated.

This version of the tau-leaping algorithm requires a computational time proportional to $2M$ (where M is the number of reactions of the system), which corresponds to the number of auxiliary quantities needed for the computation of the τ value. Note that, this τ selection strategy is faster than that of the original tau-leaping algorithm [77], whereby M^2 auxiliary quantities needed to be computed. Moreover, the tau-leaping procedure presented in this section is also faster than the SSA and the NRM, because it executes larger steps in which several reactions are applied, thus speeding up the simulations (as reported in [77, 26, 76]).

The tau-leaping algorithm has been used, for instance, to investigate the cell cycle of the unicellular budding yeast *Saccharomyces cerevisiae* [1], and to study a model that describes the expression of LacZ and LacY genes and activity of LacZ and LacY proteins in *E. coli*.

1.4 The next subvolume method

In this section, an algorithm called *next subvolume method* (NSM) [57], will be introduced. This procedure is suitable for the description of the dynamics of systems whose geometry is taken into account, and both reactive and diffusive processes are modelled.

This algorithm has been developed starting from the basic procedures introduced to give an exact Monte Carlo realisation of the CME describing homogeneous systems enclosed in a single volume (in particular, the NRM). Note that, a system can be considered homogeneous (that is, the well stirred

assumption in SSA, presented in Section 1.1), only if the time scale of the diffusive processes is much more faster than the time scale of the reactive processes.

The aim of the NSM is to propose an algorithm for the description of systems where the diffusion plays an important role for the system dynamics, e.g. the living cell. As a matter of fact, many cellular processes depend on the spatial heterogeneity [182], as, for instance, the cell division [90]. In order to describe the spatial heterogeneity, the NSM divides the reaction volume in a number of subvolumes and exploits the *reaction-diffusion master equation* (RDME) to describe the behaviour of the entire system [68]. The state of the system is characterised by the amounts of the molecular species occurring within each subvolume, whose dimension is chosen small enough to be considered homogeneous (well stirred). The diffusive processes among subvolumes are described by means of first-order reactions which represent the "movement" of molecules between adjacent subvolumes. The rate constant associated to these processes is D/l^2, where D is the diffusion constant of a particular molecular species, and l is the side length of the subvolume (whose shape is considered cubic).

Note that, in the case of biochemical systems modelled using a 3D structure, in which chemical processes are faster than diffusive ones, even if the subvolumes used in the RDME are small, the number of molecules occurring within them is usually high, in order to ensure the homogeneity. Therefore, it is not possible to use the SSA inside each subvolume, in order to apply a reaction within a single volume, at each iteration. Indeed, the computational time required for the simulation would be prohibitive, as it increases linearly with the number of subvolumes used in the system description.

On the other hand, the NSM is an efficient algorithm for the description of 3D biochemical systems, as it can sample trajectories of the Markov process associated to the RDME. Moreover, the equivalence between the trajectories obtained by means of NSM and those sampled by SSA, has been proven [57]. Hence, the NSM is an exact algorithm capable of describing the dynamics of heterogeneous systems, designed according to the RDME, and its efficiency (with respect to SSA) is due to the time needed for a computation, which increases logarithmically, rather than linearly, with the number of subvolumes.

The improvements in the performance of this algorithm rely entirely on the application of the direct method [74], to compute the time for the next reaction or diffusion event within each subvolume, combined with the strategy used in the NRM to keep track of the subvolume where the next event will occur. Stated in other words, the direct method is used to manage the internal state of the subvolume and to compute the auxiliary values needed for the computation of the time step τ, and to identify the reaction or the diffusion event to execute. On the other hand, the queue of the subvolumes, the ordering and the update processes are managed using the

same data structures and operations introduced in the NRM (the indexed priority queue implemented as a binary tree together with the indexing scheme, see Section 1.2 and [72] for additional details). Note that, during each iteration of the NSM, only one subvolume (for reaction events) or two subvolumes (for diffusion events) need to be updated, because their internal state changes.

Exploiting the strategy briefly presented above, the NSM, describing an exact numerical realization of heterogeneous systems composed by a number of subvolumes, can be formalised as follows [57]. In the following description, the algorithm execution naturally proceeds according to the order of instructions, when not otherwise specified by means of "go to" commands.

Initialisation:

1. Load the information about the geometry of the system, i.e., the number of subvolumes and their connections. Clearly, if the subvolumes have cubic shape, then they can have at most six connections to the other subvolumes;

2. Set the initial amounts for the molecular species inside each subvolume. Note that, the initial state can be assigned to the subvolumes either randomly or by using any initial distribution;

3. Compute the sum of the reaction rates r_i for each subvolume i. The rates of the reactions are calculated according to the size of the subvolume, as stated in the RDME;

4. Compute the sum of the diffusion rates for each subvolume i as $s_i = f_i \cdot \sum_{n=1}^{N}(d_n \cdot X_{n,i})$, where $d_n = D_n/l^2$ is the diffusion constant associated to the species S_n, and $X_{n,i}$ is the molecular amount of the species S_n occurring inside the subvolume i. f_i is the number of connections of the subvolume i, that is, the number of directions where the molecule can diffuse to;

5. For each subvolume i:

   a. Compute the sum of reaction and diffusion rates: $r_i + s_i$,

   b. Draw a random number rnd_{i_1} from the unit uniform distribution $[0, 1]$,

   c. Calculate the time for the next reaction as $\tau_i = 1/(r_i+s_i)\cdot 1/rnd_{i_1}$;

6. Order the subvolumes according to the values of τ_i. In a similar manner to the NRM, the subvolume queue is stored in a binary tree, indeed the subvolume with the least time associated is in the root of the tree and the branches have increasing τ value;

Iteration:

7. Select the subvolume i with the least time τ_i, the next event will occur at time $t = \tau_i$. Generate a random number rnd_{i_2} from the unit uniform distribution $[0, 1]$, if $rnd_{i_2} < r_i/(r_i + s_i)$, then go to *Step 8* to handle a reaction event, else go to *Step 9* to handle a diffusion event;

8. Reaction event:

   a. Using the random number rnd_{i_2} extracted during *Step 7*, determine the reaction to execute (as in Gillespie's direct method),

   b. Update the internal state of the subvolume i by changing the amounts of the molecular species involved in the reaction,

   c. Calculate the new values of r_i, s_i and their sum (inside the subvolume i), and compute the new value of the time for the next event $\tau_i = 1/(r_i + s_i) \cdot 1/rnd_{i_2} + t$,

   d. Insert the new τ_i value in the subvolume queue and reorder it;

9. Diffusion event:

   a. Using the random number rnd_{i_2} extracted during *Step 7*, determine the type of molecule that will diffuse. Clearly, the number of molecules that diffuse depends on the amounts of molecules (currently present within the subvolume) and on the diffusion rate constants,

   b. Choose (randomly) the direction of the diffusion event among the connection of the subvolume (and using again the random number rnd_{i_2}),

   c. Update the internal state of both the subvolume i and its neighbour j (where the molecule is sent), by changing the amounts of the molecular species involved in the diffusion event,

   d. Calculate the new values of r_i, s_i and r_j, s_j and their sum (inside the subvolume i and j), and compute the new value of the time for the next events as in *Step 8c*,

   e. Insert the new τ values in the subvolume queue and reorder it;

10. If the termination criterion is satisfied, then end the simulation. Otherwise go back to *Step 7*.

The algorithm presented above takes a computational time that is proportional to the number of reactions, to identify the time and the event which will occur next inside every subvolume (this is due to the application of the direct method). On the other hand, the subvolume queue is managed with a time proportional to the logarithm of the number of subvolumes, i.e.,

the time needed to re-order the queue is proportional to the logarithm of the number of subvolumes.

A different implementation of the NSM could be used to speed up the procedure, at the cost of using a larger amount of memory. The idea is to execute the operations within the subvolumes by exploiting the NRM. So doing, the computational time needed to handle each subvolume is proportional to the logarithm of the number of reactions. On the contrary, instead of storing only the time τ for the next event, all the candidate values of τ need to be stored. Therefore, the amount of memory required for this alternative implementation is proportional to the number of events (for each subvolume).

The NSM has been used, for instance, for the simulation of the stochastic reaction–diffusion kinetics of a double–negative feedback system, and a MAPK phosphorylation–dephosphorylation system [57].

1.5 The binomial tau-leap spatial stochastic simulation algorithm

The binomial tau-leap spatial stochastic simulation algorithm (Bτ-SSSA) has been introduced in [117], as an alternative procedure to the next subvolume method for the simulation of heterogeneous systems. The Bτ-SSSA follows the principles introduced in the NSM, but the algorithm has been modified in order to simulate the behaviour of the subvolumes following the the binomial tau-leap algorithm [196].

In this section, the binomial tau-leap will be briefly recalled, emphasizing the differences with Gillespie's tau-leaping (in particular, the version introduced in [77] will be considered here); afterwards, the Bτ-SSSA will be described.

One of the main problems of the first version of tau-leaping algorithm, as proposed in [77], concerns the robustness of the method, because the selection and execution of a number of reactions could lead to negative amounts of the molecular species occurring inside the system. To be more precise, during a tau-leaping step (which is usually larger than a SSA step), if the number of reactions to fire is greater than the number of molecules of the current system configuration, then negative populations are generated, and the simulation step cannot be executed. The solution proposed to avoid this possibility consists in undoing the last step, reducing the length of the step by half, and selecting a new set of reactions to execute. It is clear that such kind of solution increases the computational cost of the algorithm, hence different but less expensive methods are needed (note that, the tau-leaping version presented in Section 1.3 uses a different strategy to efficiently select both the time step and the rules, avoiding the negative population problem).

The method proposed in the binomial tau-leap to select the leap length

and reactions consists in the exploitation of binomial random variables rather than Poisson random variables. This different sampling is suitable for the tau-leaping because, while samples from Poisson random variables range from zero to infinity, binomial random variables have a finite range. The binomial tau-leap method, besides providing a different sampling, implements a robust control strategy to avoid negative molecular amounts.

The procedure used to tackle the problem related to negative numbers consists in bounding the sampling range of the binomial random variables according to the system state. In particular, the number of execution of a given reaction R_j, during the current iteration, is defined by a sample value of the binomial random variable $B(N_j, a_j(\mathbf{x})/N_j)$ under the following condition:

$$0 \leq \frac{a_j(\mathbf{x})\tau}{N_j} \leq 1,$$

where the value N_j is computed according to the kind of reaction. For instance, given a first-order reaction $R_1 : S_1 \xrightarrow{c_1} S_2$ (whose propensity function is $a_1(\mathbf{x}) = c_1 \cdot X_1$), the value $N_1 = X_1$. Dealing with second-order reactions, in the heterologous case, $R_2 : S_1 + S_2 \xrightarrow{c_2} S_3$ (where $a_2(\mathbf{x}) = c_2 \cdot X_1 \cdot X_2$) the value $N_2 = min\{X_1, X_2\}$. In the homologous case of second-order reactions, such as $R_3 : S_1 + S_1 \xrightarrow{c_3} S_2$ (having $a_3(\mathbf{x}) = c_3 \cdot \frac{X_1(X_1-1)}{2}$), the reaction is executable if $X_1 \geq 2$ and $N_3 = \lfloor X_1/2 \rfloor$. It is clear that, for each reaction R_j, the number N_j is used to ensure the applicability of the reaction itself, given the current system state $\mathbf{x}$.

The limitation on the number of reactions execution explained above is not enough to avoid the problem related to the negative molecular numbers if a species is involved in more than one reaction. Therefore, a different sampling method (based on binomial random variables), has been proposed. This novel strategy is founded on two properties of the Poisson and binomial random variables [196], which are exploited to limit the total number of executions of the reactions that modify the molecular quantity of a particular species. This sampling method can be explained considering two reactions R_j and R_k which modify the amount of species S_i (note that, this procedure can be easily extended to higher numbers of simultaneous reactions).

Following the (Poisson) tau-leaping procedure [77], the number of executions for reactions R_j and R_k in $[t, t+\tau)$ would be sampled from the Poisson random variables $P_j = P(a_j(\mathbf{x})\tau)$ and $P_k = P(a_k(\mathbf{x})\tau)$, respectively. On the other hand, using the binomial tau-leap, and exploiting the first property of Poisson and binomial random variables, the total number of reactions R_j and R_k to execute during the leap is first generated from the binomial random variable:

$$B\left(N_i, \frac{a_j(\mathbf{x})}{a_j(\mathbf{x}) + a_k(\mathbf{x})}\tau\right) \tag{1.15}$$

under the following conditions:

$$N_i = min\{N_j, N_k\}, \quad 0 \leq \frac{a_j(\mathbf{x}) + a_k(\mathbf{x})}{N_i}\tau \leq 1,$$

where the values N_j and N_k are computed as illustrated above, according to the X_i quantity in the state $\mathbf{x}$.

Afterwards, a second sample K_j, which corresponds to the number of executions of reaction R_j can be generated, exploiting the second property of Poisson and binomial random variables, from a distribution which depends on the value K_{jk}, that is, the total number of executions of reactions R_j and R_k:

$$B\left(K_{jk}, \frac{a_j(\mathbf{x})}{a_j(\mathbf{x}) + a_k(\mathbf{x})}\right).$$

Finally, the number of executions of reaction R_k is computed as $K_k = K_{jk} - K_j$.

This strategy is exploited in the Bτ-SSSA procedure, whenever a tau-leaping step is executed in one of the subvolumes of the system.

The Bτ-SSSA has been developed in order to speed up the NSM, applying several reaction and diffusive events at each step. During each iteration of the algorithm, a subvolume is selected according to its state (hence, to the probability values of the corresponding reactions) and the procedure to evolve its state is executed. The first operation consists in computing τ: if this value is greater than a specified threshold, then the Bτ-SSSA procedure is applied; otherwise, a modified version of the NSM is performed (executing a single event).

The Bτ-SSSA, where the probabilities associated to both reaction and diffusion events are calculated as described in Section 1.4, can be formalised as follows [117]. In the following description, the algorithm execution naturally proceeds according to the order of instructions, when not otherwise specified by means of "go to" commands.

Initialisation:

1. Load the information about the geometry of the system, i.e., the number of subvolumes and their connections;

2. Set the initial amounts for the molecular species inside each subvolume;

3. Compute the sum of the reaction rates r_i and the sum of the diffusion rates s_i within each subvolume i;

4. For each subvolume i, compute the value τ_i in which the next set of events will be executed;

5. Order the subvolumes according to the values of τ_i;

   Iteration:

6. Select the subvolume i with the least τ_i value;

7. If the total number of molecules occurring inside the subvolume is larger than 2, then go to *Step 8*. Otherwise proceed with a single event iteration step:

   a. If $r_i = 0$, then go to *Step 11* to execute a single diffusion event;
   b. If $r_i > 0$, then draw a random number rnd_1. If $rnd_1 < r_i/(r_i+s_i)$, then go to *Step 10* to execute a reaction event. Otherwise go to *Step 11* to execute a diffusion event;

8. Calculate an initial τ_i for the subvolume, as described in [196];

9. Determine whether the step consists of one event or a set of events. In the case of $r_i > 0$, if τ_i is smaller than a specified threshold, then go to *Step 7b*. Otherwise go to *Step 12*;

10. Single chemical reaction event:

    a. Draw a random number from the uniform interval $[0, 1]$ to select which reaction will be executed (following the Gillespie's method [74]);
    b. Update the molecular quantities inside the subvolume i according to the stoichiometry of the reaction executed, and increase the simulation time;
    c. Update the values of r_i and s_i inside the subvolume i;
    d. Generate a new τ_i value for the next event inside volume i and sort the subvolume queue;
    e. If the termination criterion is satisfied, then end the simulation. Otherwise go back to *Step 6*.

11. Single diffusion event:

    a. Draw a random number from the uniform interval $[0, 1]$ to select which species will diffuse to a neighbour subvolume (following the NSM method [57]);

b. Draw a random number from the uniform interval $[0, 1]$ to select the neighbour subvolume j where the molecule will diffuse;

c. Update the molecular quantities inside the subvolumes i and j according to the executed diffusion event, and increase the simulation time;

d. Update the values of the reaction and diffusion events probabilities inside the subvolumes i and j;

e. Generate new τ_i and τ_j values for the next event inside volume i and j and sort the subvolume queue;

f. If the termination criterion is satisfied, then end the simulation. Otherwise go back to *Step 6*.

12. τ-leap step:

a. Select the reaction and diffusion events according to the strategy of the binomial tau-leap presented above [196];

b. Update the system state according to the set of selected reaction and diffusion events, and update the system time;

c. Calculate the new values of the propensity functions in the subvolumes affected by the executed events;

d. Generate new τ values and sort the subvolume queue;

e. If the termination criterion is satisfied, then end the simulation. Otherwise go back to *Step 6*.

As in the first version of Gillespie's tau-leaping algorithm [77], the Bτ-SSSA needs the evaluation of M^2 auxiliary quantities to compute the length of the step τ, inside the selected subvolume. Exploiting a modified version of the NSM to manage the subvolumes, the algorithm presented here inherits the same computational time (described in Section 1.4). Comparing the number of operations needed to execute one iteration of the NSM and the Bτ-SSSA, it turns out that the latter has a higher complexity. On the other hand, the Bτ-SSSA executes a set of reaction and diffusion events (at each step), instead of a single event, increasing the efficiency of the simulations. In particular, in [117] it is reported that the Bτ-SSSA is from 2 to 100 times faster than NSM (depending on the number of subvolumes involved in the modelled system), in the examples presented by the authors.

The Bτ-SSSA has been exploited for simulating a simplified model of progesterone transcription factor formation in the epidermal growth factor receptor pathway using different numbers of subvolumes in the system's definition [117].

1.6 Other algorithms

In the last years, many other stochastic algorithms have been proposed for the simulation of biochemical systems. In this section, we recall some of these works giving a brief explanation of their features.

R-leaping. An algorithm called R-leaping [7] has been proposed for the acceleration of the SSA, by executing a predefined number of reaction firings which may occur across several reactions, with the aim to avoid the problem related to the possible occurrence of negative populations of molecules (which is present in most of the tau-leaping methods [77, 35, 196]). At each iteration of the R-leaping algorithm, assuming that the values of the propensity functions remain constant during the leap (as for tau-leaping [77]), a value representing the total number of reactions firings is computed. This value is used to extract the number of execution of each reaction by means of binomial distribution samples.

The authors demonstrate that, by periodically sorting the reactions, such that the propensity functions result in decreasing order, it is possible to minimize the number of sampling from the binomial distributions, without affecting the accuracy of dynamics described by the algorithm.

The bound on the total number of reaction executions is given as a parameter of the algorithm. Anyway, the strict limit can be relaxed in order to simulate larger steps, allowing the occurrence of negative amounts according to a controlled probability value. This relaxation strategy offers a tuning mechanism to balance accuracy and efficiency of the R-leaping according to the system state.

The R-leaping algorithm is approximately 10 times faster than SSA [74], and has the same accuracy level of tau-leaping [26], as reported in [7]; however, the computation time required for the execution of the R-leaping is greater than that needed by the tau-leaping.

The R-leaping has been applied to a LacZ/LacY model composed by 22 reactions providing good results from the simulations.

Stochastic simulation of coupled reaction–diffusion processes. The classic stochastic algorithms have been developed for the simulation of non linear chemical reaction processes in well stirred homogeneous systems. In [189], one of the first stochastic approaches which consider diffusion for non linear reaction–diffusion processes has been introduced. In order to handle diffusion processes, mesoscopic rates used to describe the transition probability associated to diffusive events have been added to the described system.

Exploiting the mesoscopic rates associated to diffusion, the algorithm provides a numerical simulation, by means of a Monte Carlo technique, of the reaction-diffusion master equation. The described dynamics considers both the spatial and temporal evolution of the molecular species occurring

inside the system. As in other methods like NSM and Bτ-SSSA, here the reaction volume is divided in subvolumes, called voxels, which are supposed to be homogeneous.

This algorithm is limited to reaction–diffusion processes where collisions between inert and reacting species occur more frequently than collisions between reacting species. Moreover, the authors do not give any insight about the computational cost of the algorithm, which seems to have a higher complexity with respect to the NSM or the Bτ-SSSA, because of the auxiliary quantities needed to compute the value of the time step τ.

This method has been applied, for instance, to the nonlinear reaction–diffusion process of calcium wave propagation through the cell, the developed model consists in 100 sequential cubic voxels since diffusion has been considered in one dimension only [189].

Next reaction hybrid algorithm. In [179], a stochastic algorithm suitable for the description of chemical systems having one or more "fast" reactions, is presented. Such kind of systems, called *stiff systems*, cannot be efficiently described by means of SSA, because the average length of the step τ might be very small, slowing down the simulation. The paper describes a hybrid stochastic method that partitions the system into subsets of fast and slow reactions. The dynamics of fast reactions can be accurately approximated as a continuous Markov process when two conditions are satisfied: (1) reactions occur many time in a small time interval, and (2) the effect of each reaction execution on the number of reactants and products is small, that is, the total amounts of those molecular species have to be large. On the other hand, the slow dynamics governed by slow reactions is described by using the next reaction method. There exists other approaches with the aim to efficiently simulate stiff systems: the slow scale SSA and the implicit tau-leaping [76].

The next reaction hybrid method has been successfully applied to several biochemical systems, such as the pulse generator, a simplified model of the molecular mechanisms responsible for the circadian rhythm in Drosophila fruit flies. However, there are two main sources of inaccuracy for this algorithm: the first one is due to the approximation of fast reactions as a continuous Markov process, in particular, this is related to the inaccuracy of the numerical integration method used (here, the Euler-Maruyama method). The authors propose the application of a more advanced stochastic numerical integrator to reduce this source of the error.

The second source of inaccuracy appears when a slow reaction is influenced by some changes in the molecular quantities of species involved in a fast reaction. In this case the source of error is related to the stochastic numerical integrator used to generate the times associated to slow reactions; again, by using more advanced methods the accuracy of the algorithm might

be improved.

Multicompartmental Gillespie's algorithm. This algorithm has been introduced in [159] to simulate systems composed by many volumes (also called compartments), exploiting the Gillespie's procedure. In particular, the direct method is used for the computation of the time τ and the index of the next reaction to execute within each compartment. This information is stored in a list which is updated at each iteration, modifying the values related to the compartments affected by the executed reaction. This strategy is very similar to that of the NSM, presented in Section 1.4, with the difference that the multicompartmental Gillespie's algorithm does not use a particular data structure such as a heap or an indexed queue to efficiently handle compartments information.

In the description of biochemical systems modelled by means of this method, the so called *boundary rules* are exploited. Such kind of reactions are used to capture the features of the communication and the transformation of molecules. In particular, the authors consider special cases of boundary rules which involve molecules occurring in different compartments, though it is not very clear how Gillespie's theory for single volume systems can be used to describe the propensity functions of reactions that are simultaneously active in two compartments, nor in which compartment this information is used to compute the value of τ.

Chapter 2

Optimization algorithms

In this chapter two optimization techniques will be presented: genetic algorithms and particle swarm optimizer. These optimisation methods can be applied in the field of modelling and simulation of biological and chemical systems since many numerical factors, like molecular quantities and reaction rates, which represent an indispensable quantitative information to perform computational investigations of the system's behaviour, are often not available or inaccurate. Optimization methods can be used to tackle the calibration problem of *parameter estimation* of these systems by minimizing a cost function (e.g. a distance measure) which quantitatively defines how good is the system behaviour using the predicted values, with respect to the experimental dynamics. In Chapter 6 the parameter estimation issue, and the application of genetic algorithms and particle swarm optimizer to this particular problem will be described.

Genetic Algorithms (GAs) [89] are one of the oldest and mostly used versions of evolutionary algorithms. The most commonly used GAs formulation evolves fixed length strings of characters from a limited alphabet, but there exists other versions where each allele contains any floating point value from a limited range. This GAs version is often called *real valued* or *real coded* GAs [204].

In Section 2.1, an introduction to the basic features of GAs is given. Afterwards, binary coded GAs and real coded GAs, along with their commonly used variation operators, are presented.

Particle swarm optimizer (PSO) [99] is a population based optimisation heuristic, inspired by the social behaviour of bird flocking or fish schooling. In PSO the potential solutions, called particles, are identified by their coordinates in the problem space and are characterized by a velocity that allows them to update their current positions. The PSO concept consists in changing, at each iteration, the velocity of each particle towards some attractors, in order to reach optimal positions in the search space.

In Section 2.2, the basic notions of PSO are explained. Moreover, an

analysis of the role of the algorithm's parameters involved in the velocity update operation is given.

2.1 Genetic algorithms

Genetic algorithms are search methods which exploit principles inspired by evolutionary theories, to the aim of evolving candidate solutions for a given problem and finding optimal solution [89, 79]. Solutions are arranged in a populations of individuals which evolve by means of competition and controlled variation. The individuals compete in a process called *selection*, which is executed according to a quality measure (fitness). Hence, individuals with higher fitness will be selected with higher probability. New individuals are then generated starting from the selected ones, by applying genetic operators called *crossover* and *mutation*, which mix the parents characteristics, and introduce new genetic material, respectively.

One of the reasons why GAs succeed in optimization problems is due to their capability to exploit the information accumulated about an initially unknown search space, and *adapt* the individuals in order to conduct successive explorations within useful subspaces. GAs are therefore suitable for large and complex search spaces, where other "classical" search approaches are inappropriate. GAs offer a valid method to optimize problems that require (computationally) efficient and effective techniques.

In this section, two GAs version will be presented: binary-coded GAs and real-coded GAs. We start with a brief overview about GAs, explaining the common features of the two versions. GAs are initialised with a population of individuals that are usually randomly generated, each one corresponding to a potential solution of the given problem. During each iteration, also called *generation*, the quality of the individuals is evaluated through a fitness function, and the individuals for the next generations are finally selected, recombined and variated. A general implementation for GAs can be formalised as follows:

1. Initialise the system by randomly generating an initial population of individuals;
2. Evaluate the fitness function for each individual;
3. Select the individuals for the next generation;
4. Recombine and vary the selected individuals;
5. If the termination criterion is satisfied, then end the procedure. Otherwise go back to *Step 2*.

The algorithm reported above represents one possible variants of GAs, however, all the implementations for these algorithms are based on the same

fundamental mechanism for the evolution of the individuals, which consists in three operations: *evaluation* of the fitness function, *selection* of the best performing individuals, and *recombination* of the characteristics of the selected individuals. Clearly, the execution of a GA terminates when some criterion is satisfied. For instance, a termination criterion can be related to the number of iteration executed, or to the best value of the fitness function obtained from an individual.

Evaluation of the fitness function. The fitness function used to evaluate the quality of each individual is independent from the implemented GAs version. On the other hand, the fitness evaluation strongly depends on the optimization problem to which the GAs are applied, and on the representation used for the individuals.

GAs are commonly applied to function optimisation problems, in which the aim is to find a set of parameter values that maximise a complex multiparameter function. Hereafter, we report a simple example where the aim is to maximise the real–values one–dimensional function

$$f(y) = y + |sin(32y)|,\ 0 \leq y < \pi.$$

The candidate solutions of this problem are the values of y, which can be encoded by means of binary strings representing real numbers. The fitness function can be defined in order to translate a given binary string x (an individual) into a real number y, and then to evaluate the function at that value. Clearly, in this case the fitness value of x corresponds to the function value evaluated at the point y.

In Section 6.3, a fitness function developed and applied to the problem of parameter estimation of biochemical systems will be described.

Selection of the best performing individuals. Given a population P of N individuals i, $i = 1, \ldots, N$, (N is also called the *size* of P), the selection procedure is used to obtain an intermediate population P' which is composed by copies of individuals from P. The number of copies of an individual added to P' is directly dependent on its fitness value. Individuals with higher fitness generally have greater chance to be selected for the intermediate population.

There exist different selection strategies, the first we present here is called *roulette wheel selection* [89, 79], a procedure where the selection of individuals is proportional to their fitness values. For each individual i in P, the probability p_i of including a copy of i to P' is computed as:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}.$$

The selection procedure can be imagined as the spinning of a roulette wheel a number of times equal to the size of the population; clearly, the number of fittest individuals added to P' will be higher than the number of the weakest individuals. So doing, individuals with fitness values above average will be more represented in the next generations, while individuals below average fitness tend to extinct.

The second selection approach recalled here is the *ranking selection* [8]. The individuals are first sorted according to their fitness values, and then the computation of p_i is based on the rank of i (rather than on its fitness value), by using a non-increasing assignment function. The ranking selection is used when a balancing (or unbalancing) between strong and weak individuals is required.

The last selection procedure here described is called *tournament selection* [131]. This kind of selection is performed by executing a number of "tournaments" equal to the size of the population. During each tournament, a few individuals are randomly selected from P, and the winner, that is, the individual with the best fitness among them, is added to the intermediate population P'. The number of participants to the tournament (k) is directly related to the selection pressure. In fact, if k is large, then weak individuals have a smaller probability to be selected for P'. Note that, if $k = 1$, the tournament selection is equivalent to a random selection of the individuals, where each individual has the same probability to be included in P'. The individuals selected by means of a tournament can be removed from P, unless it is feasible that the same individuals is allowed to participate to distinct tournaments and be selected many times for the intermediate population. The tournament selection has several advantages: (1) the fitness evaluation of the entire population is not required, indeed, only the individuals selected for the tournaments are evaluated; (2) it is easy to code and can be executed on parallel architectures; (3) the selection pressure can be easily adjusted (by changing the value of k).

Recombination of the characteristics of the selected individuals. After the construction of the intermediate population P' by means of any selection procedure, the variation operators can be applied in order to obtain new individuals (offspring) for the next generation.

The first operator applied is called *crossover*, it is used to mix the information between individuals. Crossover combines the features of two parent individuals, with the aim to obtain two offspring with better characteristics. This operator plays a central role in GAs, and is considered one of the most characterising feature of this optimization technique. In general, crossover is not applied to all selected pairs of individuals: its application depends on a probability value p_c, called crossover rate. Moreover, different versions of crossover can be defined according to the representation chosen for the

individuals.

The second operator is called *mutation*, whose aim is to arbitrarily alter one or more components (allele) of a selected individual, in order to increase the variability inside the new generation. Mutation introduces new genetic material into the population, preventing the convergence of the individuals to local optimal solutions, and it ensures that the probability to reach every point in the search space is always greater than zero. Similarly to crossover, the mutation operator is applied with a probability p_m, called mutation rate; hence, a randomly selected allele of each individual undergoes a change of its value according to p_m.

Typically, the parameter setting for these operators is chosen such that crossover is applied with high probability values (for instance, 0.95), and mutation is executed with very low rates (like 0.05), in order to reflect the natural evolution process where usually the characteristics of two individuals are mixed through, e.g., sexual reproduction, while mutations happen with very low probability.

The application of these variation operators might lead to the extinction of the best individual of the population, either because it gets mixed with another individual or because of the mutation of one of its allele. Hence, after the construction of the intermediate population P', it is possible to apply an additional selection strategy, called *elitism* [46], to "copy" the best individual (or a subset of best individuals) into the population of the next generation without modifying it.

Binary–coded GAs

The first version of GAs, which we present now, exploits a binary coding to represent the individuals of the population. Given the search space S, the points in S are codified by means of strings over the alphabet $V = \{0, 1\}$, having fixed-length L. For instance, a GA with population size equal to 5 and individual length equal to 7, can be randomly initialised as:

$i_1 = 0100101$
$i_2 = 1101010$
$i_3 = 1111000$
$i_4 = 0100110$
$i_5 = 1000000$

As previously explained, after the fitness evaluation of the individuals and the selection procedure to obtain the intermediate population P', crossover and mutation are applied.

The classical crossover operator for binary coded GAs is named *simple crossover* [89, 79]. Given two individuals $i_j = (a_j^1, \ldots, a_j^L)$ and $i_k = (a_k^1, \ldots, a_k^L)$ randomly chosen from P', two offspring $i'_j = (a_j^1, \ldots, a_j^c, a_k^{c+1}, \ldots, a_k^L)$ and $i'_k = (a_k^1, \ldots, a_k^c, a_j^{c+1}, \ldots, a_j^L)$ are generated. The crossover

point c is randomly selected from the uniform interval $[1, \ldots, L-1]$.

There exist different kinds of crossover. Among the most important ones, we recall: (1) *n-points crossover* [60], which represents a generalisation of simple crossover, since n different crossover points are randomly selected and the resulting segments of the parents individuals are exchanged to generate the offspring; (2) *uniform crossover* [190], where the value of each allele in the offspring is determined by the uniform random choice between the values of the corresponding allele in the parents.

The *mutation* operator [89, 79] is applied to an individual by randomly choosing an allele and swapping its value (from 1 to 0 or vice versa).

Many advantages are given by the use of binary alphabets in GAs. First of all, this alphabet maximizes the level of *implicit parallelism*, as explained in [80]. This property is related to the useful *implicit* information processed "in parallel" by GAs, such as schemata (besides the *explicit* information represented by the individuals). Moreover, the use of a binary alphabet provides a benefit in terms of computational cost. Indeed, when alphabets with higher cardinality are used, a larger population size is often needed in order to well represent every character [172], thus resulting in a loss of efficiency (especially when the fitness function computation is expensive).

On the other hand, the binary representation of individuals is not suitable when dealing with a continuous complex search space with large dimensions, and when a high numerical precision is required. Furthermore, when an individual can only assume a finite number of discrete valid values (which is not a power of 2), some of the binary values are redundant.

Real–coded GAs

The second version of GAs briefly described here utilises a real coding for the individuals. In this case, each allele is codified as a real number which represents a variable of an optimization problem with values in continuous domains. Therefore, the individual is implemented as a vector of floating point numbers, whose precision is related to that of the computer with which the execution of the algorithm is performed. The size of the individuals is equal to the length of the vector representing a candidate solution to the analysed problem, hence each allele codifies a variable. The values of the allele are forced to range in the intervals of the corresponding variables, and also the the values resulting from the application of variation operators have to follow this requirement.

Given two individuals $i_j = (a_j^1, \ldots, a_j^L)$ and $i_k = (a_k^1, \ldots, a_k^L)$, which have been randomly selected from the intermediate population P', the effect of the most used kinds of crossover will be described in the following. Note that, each crossover operator generates a different number of offspring, therefore a strategy called *offspring selection mechanism*, for deciding which ones will be included in the next generation, is used in some cases.

The first operator is called *flat crossover* [169], it generates an offspring $i_o = (a_o^1, \ldots, a_o^L)$ where the values a_o^l $(l = 1, \ldots, L)$ are randomly chosen from the uniform interval $[a_j^l, a_k^l]$.

Using the *simple crossover* [204, 129], a position $c \in \{1, 2, \ldots, L-1\}$ is randomly chosen and two offspring $i_{o_1} = (a_{o_1}^1, \ldots, a_{o_1}^c, a_{o_2}^{c+1}, \ldots, a_{o_2}^L)$ and $i_{o_2} = (a_{o_2}^1, \ldots, a_{o_2}^c, a_{o_1}^{c+1}, \ldots, a_{o_1}^L)$ are obtained.

With the *arithmetical crossover* [129], two offspring $i_{o_h} = (a_{o_h}^1, \ldots, a_{o_h}^L)$ $(h = 1, 2)$ are generated, by computing the values of the allele as $a_{o_1}^l = \lambda a_j^l + (1-\lambda) a_k^l$ and $a_{o_2}^l = \lambda a_k^l + (1-\lambda) a_j^l$ $(l = 1, \ldots, L)$. λ is a used–defined constant (uniform arithmetical crossover) or is updated according to the current generation (non-uniform arithmetical crossover).

Applying the *BLX-α crossover* [61], an offspring $i_o = (a_o^1, \ldots, a_o^c, \ldots, a_o^L)$ is generated; the value a_o^c is a random number sampled from the uniform interval $[a_{min} - I \cdot \alpha, a_{max} + I \cdot \alpha]$, where $a_{max} = max\{a_j^c, a_k^c\}$, $a_{min} = min\{a_j^c, a_k^c\}$ and $I = a_{max} - a_{min}$.

In the *linear crossover* [204], three offspring $i_{o_h} = (a_{o_h}^1, \ldots, a_{o_h}^L)$ $(h = 1, 2, 3)$ are obtained, where $a_{o_1}^l = \frac{1}{2} a_j^l + \frac{1}{2} a_k^l$, $a_{o_2}^l = \frac{3}{2} a_j^l - \frac{1}{2} a_k^l$ and $a_{o_3}^l = -\frac{1}{2} a_j^l + \frac{3}{2} a_k^l$. In this case, the offspring selection mechanism is applied with the aim to select the two most promising offspring.

The operator called *discrete crossover* [143] generates an offspring $i_o = (a_o^1, \ldots, a_o^L)$ whose elements a_o^l are randomly chosen from the set $\{a_j^l, a_k^l\}$.

Using the *extended line crossover* [143], an offspring $i_o = (a_o^1, \ldots, a_o^c, \ldots, a_o^L)$ having elements $a_o^l = a_j^l + \alpha(a_k^l - a_j^l)$, is generated. The value of α is randomly chosen from the uniform interval $[-0.25, 1.25]$.

The *extended intermediate crossover* [143] is slightly different from the previous one, here an offspring $i_o = (a_o^1, \ldots, a_o^c, \ldots, a_o^L)$ having elements $a_o^l = a_j^l + \alpha^l(a_k^l - a_j^l)$, is generated. The value of α^l is randomly chosen (for each allele) from the uniform interval $[-0.25, 1.25]$.

The last crossover operator we recall here is the *Wright's heuristic crossover* [204]. Supposing that i_j is the parent with the best fitness value, then the values of the offspring are computed as $a_o^l = r \cdot (i_j^l - i_k^l) + i_j^l$, where r is a random number sampled from the interval $[0, 1]$.

After the application of the crossover operator on the intermediate population P', the mutation operator is applied. Given an individual $i = (a^1, \ldots, a^L)$, the allele a^l (whose value ranges in the interval $[a_{min}^l, a_{max}^l]$) is the value randomly selected that will be mutated, and $\bar{a}^l$ is the value of the allele after the application of a mutation operator. In what follows, the effect of different mutation operators will be shown.

Using *random mutation* [129], $\bar{a}^l$ is obtained as a random number drawn from the uniform interval $[a_{min}^l, a_{max}^l]$.

Applying the *non-uniform mutation* [129] in the generation t, with t_{max} as the maximum number of generations, then:

$$\bar{a}^l = \begin{cases} a^l + \Delta(t, a^l_{max} - a^l) & \text{if } \tau = 0 \\ a^l - \Delta(t, a^l - a^l_{min}) & \text{if } \tau = 1 \end{cases}$$

where τ is a random number whose value can be either 0 or 1, and

$$\Delta(t, y) = y(1 - r^{(1-\frac{t}{t_{max}})^b}),$$

where r is a random number sampled from the interval $[0, 1]$, and b is a user–defined parameter which determines how much mutation is dependent on the number of iterations. The Δ function gives a number in the range $[0, y]$ such that the probability of returning a number close to zero increases with the number of iterations. Therefore, using this operator the mutation is "strong" during the first iterations, achieving a wide range search; conversely, in the later stage of the algorithm, the mutation is "weak", resulting in a local tuning of the value.

Another mutation operator is called *real number creep* [44]. This strategy is applied when an individual is in a "good" local maximum, and the exploration of its neighbourhood might be interesting. In order to achieve this, the allele is increased or decreased by a small random quantity sampled from a user defined range.

The *Mühlenbein's mutation* [143] changes the value of the allele as $\bar{a}^l = a^l \pm rang^l \cdot \gamma$, where $rang^l$ indicates the mutation range (and it is usually set to $0.1 \cdot (a^l_{max} - a^l_{min})$), and $\gamma = \sum_{j=0}^{15} \alpha_j \cdot 2^{-j}$ (where 15 represent the precision of the operator). The value of $\alpha \in \{0, 1\}$ is randomly generated with $p(\alpha = 1) = \frac{1}{16}$. The value of the allele is increased or decreased with probability 0.5, and ranges in the interval $[a^l - rang^l, a^l + rang^l]$ with a precision of $rang^l \cdot 2^{-15}$ for the smallest variation.

There exist other crossover and mutation operators for real coded GAs, which are generalisations of the operators presented above or other novel strategies suitable for convex search spaces. An overview of these methods can be found in [88].

The advantages of using real coded GAs are many, for instance, they allow to handle large domains for the involved variables, which is instead difficult to achieve by using a binary implementation, where increasing the domains results in a loss of precision (assuming fixed length individuals). Moreover, the real coding allows a local tuning of the solutions, since small variations in the variable values result in slight changes in the (entire) individual. On the contrary, in the binary coding, a swap of a single allele can produce a completely different individual. Finally, the real coding representation of the solution is close to the "natural" formulation of many problems, hence the encoding and decoding processes, which are necessary for binary GAs versions, can be avoided, increasing the algorithm efficiency.

GAs have been successfully applied to a wide variety of applications. For instance, GAs can be applied to the field of control systems engineering; indeed, in most of the controller designs, some parameters are required to be optimized in order to give a better overall control performance. Furthermore, the configuration or the order of the controller can be optimized to reduce the system's complexity. Another example of GAs application regards the patterns recognition. GAs can be applied to generate the image filters for a two–stage target recognition system in such a way that the target image from background clutter is vividly classified from the imaging data. GAs can be used in automatic speech recognition systems where the spoken speech patterns (test pattern) are usually identified by using the pre-stored speech patterns (reference patterns). The comparison of speech signals has a number of difficulties as variations in time and the time scales among them are not fixed. Therefore, time registration of the test and the reference patterns is one of the fundamental problems in the area of automatic isolated word recognition. Finally, GAs can be applied to protein folding simulations problem. Here, the population consists in conformations of the polypeptide chain. Then, conformations are changed by mutation, in the form of conventional Monte Carlo steps, and crossover is executed by interchanging parts of the polypeptide chain between conformations.

We refer the reader to [113, 200] (and references therein) for additional information about GAs applications.

2.2 Particle swarm optimizer

Particle swarm optimizer (PSO) [99] is a method for the optimization of continuous nonlinear functions, which has been developed simulating simplified social behaviours [174], taking inspiration from herds, schools and flocks. The theory behind PSO states that individual members of a school can profit from the discoveries and previous experience of all other members during the search of food. This advantage can become decisive, outweighing the disadvantages of competition for food items, whenever the resource is unpredictably distributed in patches. In other words, this statement indicates that social sharing of useful knowledge among members of the same species represents a fundamental evolutionary advantage.

In order to simulate the search for food, the members of the swarm (called particles) need to be moved within the search space. The movement is accomplished according to two properties: nearest-neighbour velocity matching and "craziness". The velocity of each member is updated according to its nearest neighbour, resulting in a coordinated movement. So doing, the swarm quickly settles on an unchanging direction. Therefore, a stochastic factor called "craziness" is used to add a sort of noise to the movement of the swarm members.

PSO has the advantage of relying on very simple concepts, making it easy to implement and computationally inexpensive. Moreover, the performances of PSO on test functions are comparable to that of GAs, but this optimization technique does not suffer from some of the difficulties of GAs: first, the interaction of the group enhances the progress towards the optimum solution (while in GAs, interaction by means of crossover can have the opposite effect). PSO has memory, in fact, particles are always attracted by the best positions found so far, because good solutions are stored. On the contrary, in GAs, changes result in destruction of previous knowledge, except when "elitism" is applied to the population.

In both PSO and GAs, the initialisation consists in a population of random solutions to the considered problem. However, PSO's particles are also characterised by a random velocity, and they can "fly" through the search space. As previously said, the particles have memory, since they keep track of the best solution they have achieved so far; this particular value is called *personal best*. Another "best" value is traced, according to the PSO version used, which can be a *global best*, that is, the best position found so far by the entire swarm, or a *local best*, namely, the best position found by the neighbourhood of the particle. The difference between the "global" and "local" version of PSO, as reported in [54], consists in the fact that using a global best information shared among the particles, the average number of iterations needed by the swarm to converge to an optimum solution is smaller. Conversely, using a local best strategy, neighbour particles groups spontaneously separate and explore different regions of the search space, making the method more resistant to local optima. The evaluation of the quality of the particles position is measured by means of a fitness function. This function is independent from the different PSO versions, but it is strictly associated to the optimisation problem to which PSO is applied.

Thanks to the information carried on by particles, PSO (and in general, swarm intelligence) follows five basic principles. First, the proximity principle, which states that the population should be able to accomplish simple space and time computations. Second, the quality principle: the population should respond to quality factors in the environment. Third, the diverse response principle: the population should not behave in a completely coordinated manner. Fourth, the stability principle: the behaviour of the population should not change every time the environment changes. Fifth, the adaptability principle: the population must change behaviour mode when it is advantageous.

To be more precise, PSO meets the five principles because, first, it consists of a series of time steps in which n–dimensional computations are accomplished. Second, the population responds to the personal and global/local best quality factors. Third, different influences from the quality factors lead to diverse responses among particles. Fourth, the population behaviour changes only when the best position changes. Fifth, the popula-

tion is adaptive, in the sense that it does change its behaviour when the best positions changes. PSO is defined as a set of N particles, which are placed in a n–dimensional space. Each coordinate of the particle $\overrightarrow{x_i}$ $(i = 1, \ldots, N)$ is denoted by means of the element $x_{i,d}$ $(d = 1, \ldots, n)$. $\overrightarrow{p_i}$ and $\overrightarrow{p_g}$ represent the personal best and the global best found so far by the particle $\overrightarrow{x_i}$, respectively. The vector $\overrightarrow{v_i}$ denotes the velocity of the particle $\overrightarrow{x_i}$ and it can range within the interval $[-V_{Max}, V_{Max}]$. rnd_1 and rnd_2 are two random numbers.

The basic version of the PSO can be formalised as follows:

1. Initialise the system assigning to each particle a random position and velocity;
2. For each particle i and for each dimension d do:
   - if fitness(i) $<$ best$_i$ then $p_{i,d} = x_{i,d}$;
   - $v_{i,d} = v_{i,d} + rnd_1(p_{i,d} - x_{i,d}) + rnd_2(p_{g,d} - x_{i,d})$;
   - if $v_{i,d} > V_{Max}$ then $v_{i,d} = V_{Max}$
     else if $v_{i,d} < -V_{Max}$ then $v_{i,d} = -V_{Max}$;
   - $x_{i,d} = x_{i,d} + v_{i,d}$;
3. If the termination criterion is satisfied, then end the procedure. Otherwise go back to *Step 2*.

During the first step, as already stated above, the population is randomly initialised in the n–dimensional search space, by assigning a random value to each coordinate $x_{i,1}, \ldots, x_{i,n}$ of the particle $\overrightarrow{x_i}$. Moreover, also the initial velocity $\overrightarrow{v_i}$ is randomly assigned to the particles.

During each iteration of the algorithm, the position of the particle i is tested, in order to quantify its quality by means of a fitness function. If the current position is better than the previous personal best position ($\overrightarrow{p_i}$), then the position $\overrightarrow{x_i}$ is stored in the vector $\overrightarrow{p_i}$.

By evaluating all the particles of the swarm, it is also possible to identify the global or local best position ($\overrightarrow{p_g}$). The global best position corresponds to the best performing particle of the entire population, while the local best position regards the neighbourhood of a particle. One of the most used concept of neighbourhood consists in considering the particles $i-1$ and $i+1$ for each particle i in the swarm.

The values rnd_1 and rnd_2 are positive random numbers drawn from the interval $[0, c_1]$ and $[0, c_2]$, respectively, which are used to update the particle velocity. c_1 and c_2 are system's parameters, whose value is fixes a priori of can vary during the execution, according to the implemented PSO version. The values of the velocity $\overrightarrow{v_i}$ of the particle $\overrightarrow{x_i}$ is limited by the value V_{Max}. V_{Max} is used because there are cases in which the particle's velocity $\overrightarrow{v_i}$ can increase without limits, at each iterative step of the algorithm. The value of

V_{Max} can be arbitrarily selected, though it is usually set according to some knowledge of the problem. Otherwise, the value of V_{Max} can be the same of X_{Max}, which is the search space limit of the particles.

The velocity of a particle is directly influenced by the distance from its personal best ($\overrightarrow{p_i} - \overrightarrow{x_i}$) and from the local/global best ($\overrightarrow{p_g} - \overrightarrow{x_i}$), and the updated value of $\overrightarrow{v_i}$ is used during the final stage of step 3 of the algorithm, to change the particle position such that $\overrightarrow{x_i}(t) = \overrightarrow{x_i}(t-1) + \overrightarrow{v_i}(t)$, thus simulating the movement in the search space toward the best positions found so far.

During the iterations of the PSO algorithm, better positions found by particles are substituted into $\overrightarrow{p}$ vectors, hence, the population converges towards optimal regions of the search space. The success of PSO is mainly due to the interactions among the particles, which influence the trajectories during the exploration of the solutions space.

It is worth noticing that the algorithm works on the values $\overrightarrow{v_i}$ used to update the position of the particles, rather than modifying the positions $\overrightarrow{x_i}$ of the particles. In particular, a particle does not move directly towards the best position, but it will tend to circle around it, because its direction is only in part influenced by the two system's attractors, namely, the personal and local/global best positions.

The execution of the PSO terminates when some criterion is satisfied. For instance, a termination criterion can be related to the number of iteration executed, or to the best value of the fitness function obtained from a particle's position.

The parameters c_1 and c_2. The trajectory of a particle depends on the c_1 and c_2 values used during the velocity update process.

As shown in [98], the trajectories that the particles will undertake during PSO iterations, can be described by analysing the following simplified system, changing the value of c:

$$\overrightarrow{v_i}(t) = \overrightarrow{v_i}(t-1) + c(\overrightarrow{p_i} + \overrightarrow{v_i}(t-1))$$
$$\overrightarrow{x_i}(t) = \overrightarrow{x_i}(t-1) + \overrightarrow{v_i}(t)$$

The initial values of $\overrightarrow{v_i}$, $\overrightarrow{p_i}$, and $\overrightarrow{x_i}$ can be arbitrarily set (with $\overrightarrow{v_i} \neq 0$). In the examples below, the system will be initialised with $\overrightarrow{x_i}(0) = \overrightarrow{p_i} = 0$ and $\overrightarrow{v_i}(0) = 2$.

The value of c has been varied within the range $[0, 4]$. With $c = 0$, the trajectory becomes $\overrightarrow{v_i}(t) = \overrightarrow{v_i}(t-1)$, thus the particle continues on its initial direction in each iteration. Changing the c value in the interval $(0, 0.5]$, the particle trajectory has a sort of sinusoidal form, as shown in Figure 2.1 (left) where $c = 0.1$. Increasing the value of the parameter c, the amplitude of the trajectory decreases, and exceeding the value 0.5, some irregularity in the

sinusoidal pattern appears (Figure 2.1 (right)). When the value of c is set to 1, the amplitude of the oscillations is equal to the initial value of $\overrightarrow{v_i}$, while, in general, in the interval $(0, 1)$ the amplitude is greater than the value of $\overrightarrow{v_i}(0)$.

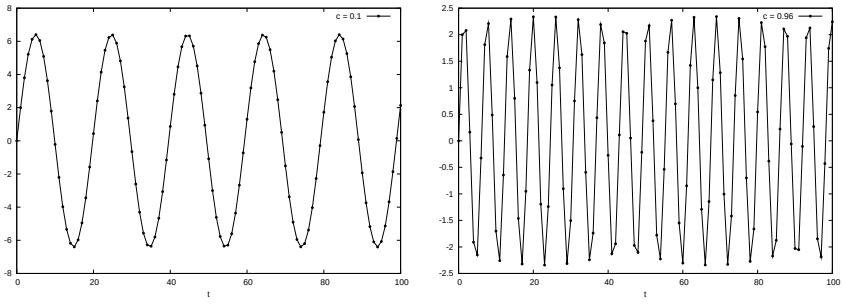


Figure 2.1: Particle trajectory with parameter $c = 0.1$ (left) and $c = 0.96$ (right).

Increasing c above 1.0 results in higher frequencies of the oscillations, and a quasi periodic pattern is visible within each oscillation of the trajectory (intra oscillation pattern, in what follows), as shown in both graphs of Figure 2.2, where c was set to 1.35 (left) and 1.9 (right). Moreover, as c increases, the length of the pattern decreases (approximatively, from 6 to 4 points per oscillation).

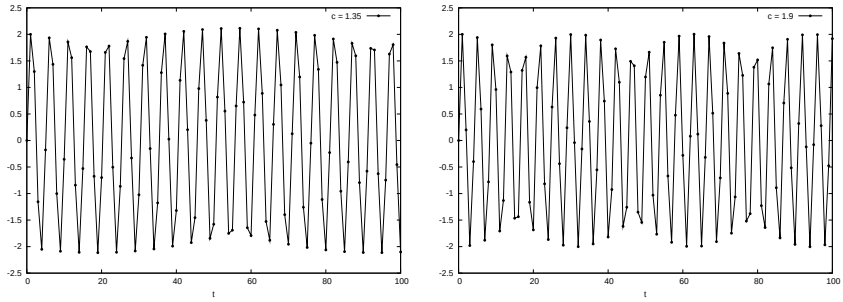


Figure 2.2: Particle trajectory with parameter $c = 1.35$ (left) and $c = 1.9$ (right).

In the interval $[2, 3]$, the length of the intra oscillation pattern becomes 3, and we pass from trajectories characterised by irregular behaviours, for instance with $c = 2.46$, as shown in Figure 2.3 (left), to quasi periodic oscillations, increasing c above 2.8. In Figure 2.3 (right), the quasi periodic trajectory of the particle with $c = 2.9$, is depicted. Note that, in the interval $[1, 3]$, the amplitude of the oscillations is always equal to the initial value of the velocity.

Increasing c over the value 3, the pattern which characterises the oscillation in the interval $[2, 3]$ is still visible. However, above $c = 3.3$ a differ-

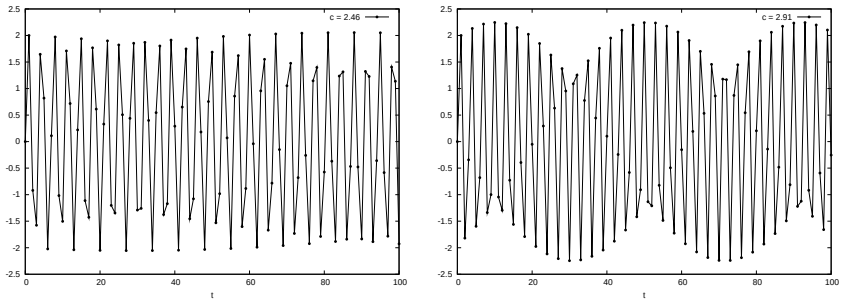


Figure 2.3: Particle trajectory with parameter $c = 2.46$ (left) and $c = 2.91$ (right).

ent pattern appears in the trajectory, as shown in Figure 2.4 (left), where $c = 3.66$. Again, over the value 3.9, a different shape of the oscillation appears, as plotted in Figure 2.4 (right), for $c = 3.99$. In general, for values of $c > 3$, the amplitude of the oscillation is greater than the value of $v(0)$.

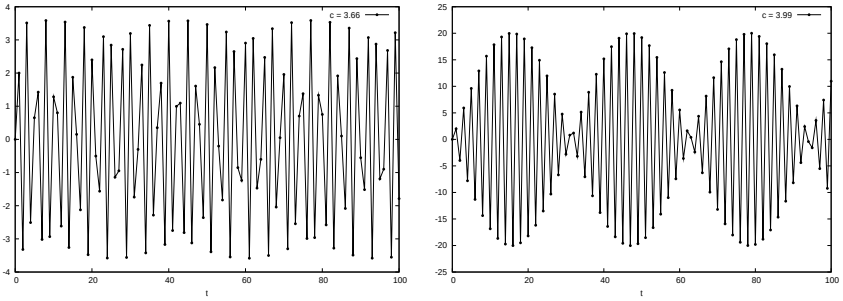


Figure 2.4: Particle trajectory with parameter $c = 3.66$ (left) and $c = 3.99$ (right).

When c reaches the value 4, the trajectory explodes linearly (Figure 2.5 (left)), while for $c > 4$, the explosion is exponential (Figure 2.5 (right)).

Summarising, the trajectory of the particles is deeply influenced by the settings of parameter c, anyhow, in [98] has been reported that the best performances of PSO can be obtained using c values around 2.

It is also possible to add a stochastic factor to the deterministic trajectories presented above. This factor is represented by a random number drawn from the unit uniform distribution $[0, 1]$, which makes the system less predictable and more flexible (Figure 2.6 (left)). In this particular case, the upper limit on the velocity value (V_{Max}) must be used because very small amounts of randomness can result in rapid explosion (Figure 2.6 (right)).

The inertia weight w**.** The problem of the unlimited increase of the particles velocity has been tackled introducing an additional parameter in

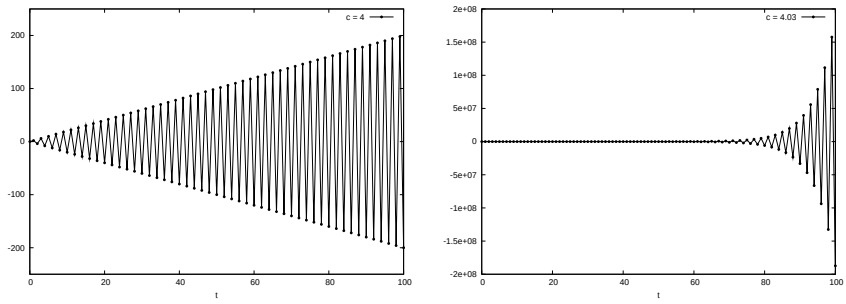


Figure 2.5: Particle trajectory with parameter $c = 4$ (left) and $c = 4.03$ (right).

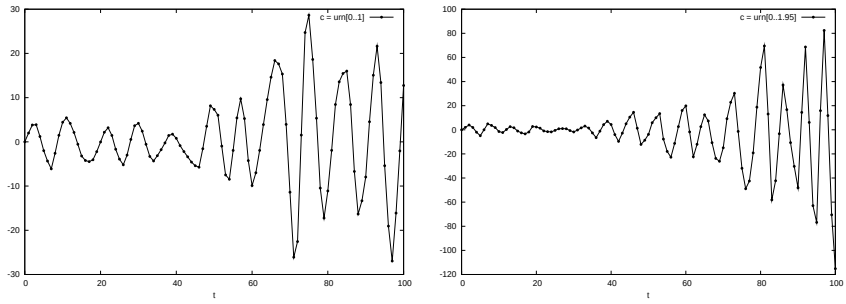


Figure 2.6: Particle trajectory with parameter c randomly drawn from the uniform interval $[0, 1]$ (left) and c randomly drawn from the uniform interval $[0, 1.95]$ (right).

the update equation used by PSO. This new term is called *inertia weight*, denoted by w, and it is used to influence the impact of the previous velocity of the particle [183]:

$$v_{id} = w \cdot v_{id} + c_1(p_{id} - x_{id}) + c_2(p_{gd} - x_{id})$$

It is worth noticing that the value of the inertia w controls the balancing between the global and local exploration ability of the particles. In fact, large inertia values lead to global exploration of the search space, meaning that the particle "flies" to new areas; on the contrary, small inertia weight values tend to facilitate local exploration, constraining the particle to the current search area (which is, therefore, better explored).

A good solution for the inertia weight setting consists in starting from a (relatively) large value and then decreasing it, at each iteration. This strategy permits the global exploration of the search space, at the beginning of the computation, and then, when the inertia weight value becomes small, the particles will focus on a small area where the current best position have been found.

In [183], the PSO algorithm has been executed on a number of test functions with different settings, the best results obtained by the authors regard a configuration in which the inertia weight has been decreased from 0.9 to 0.4 during the first 1500 iterations, and then kept constant until the end of the simulation.

Instead of a simple linear variation for inertia weight, in [34] a different approach, which consists in a non-linear decrease of the inertia weight, has been proposed. Using this strategy, the value of w is updated, at each iteration, according to the following equation:

$$w(t) = \left(\frac{(t_{max} - t)^n}{t_{max}^n} \right) (w_{initial} - w_{final}) + w_{final}$$

where t_{max} is the given maximum number of iterations of the PSO execution, $w_{initial}$ and w_{final} are the initial and final value of the inertia weight, respectively, and n is the nonlinear modulation index. Note that, if $n = 1$, then the inertia weight value is decreased linearly.

The last PSO version that we recall here, has been introduced in [135]. In this work the authors present a novel update strategy. In particular, they add "noise" to the c parameters and to the global best position used to update the particles velocity. At each iteration, the parameters undergo a "mutation" of their values, namely, a random number drawn from a Gaussian distribution $N(0, 1)$ with mean 0 and standard deviation 1 is added to their current values.

PSO has been successfully applied to a wide range of applications. For instance, this technique has been used to evolve artificial neural networks, by optimising both the weights and the network structure. An example of PSO applied to a neural network training is related to the analysis of human tremor. The diagnosis of human tremor (such as Parkinson's disease) is a very challenging area; PSO has been used to evolve a neural network used to distinguish between normal subjects and those affected by tremor. Another example of PSO application regards the optimisation of reactive power and voltage control systems. Here, PSO has been used to determine a control strategy with continuous and discrete variables, resulting in a hybrid version of the algorithm. Finally, PSO has been employed to optimise the ingredients mixture used to grow production strains of microorganisms. In particular, PSO was used together with other classic optimisation methods, but the fitness values obtained by using PSO were twice the values provided by classic methods.

We refer the reader to [55] (and references therein) for additional details about PSO applications.

Chapter 3

Membrane systems

In this chapter some prerequisites and basic notions of *membrane systems* will be recalled. Membrane systems, also called P systems, have been introduced as a class of parallel, distributed and nondeterministic computing devices, inspired by the architecture and the functioning of living cells. Many variants of membrane systems have been presented to provide a suitable modelling framework for chemical, biological and ecological systems [164, 31, 16, 17]. P systems provide a suitable modelling framework for biological and chemical systems composed of many volumes. Within the membrane defined in the structure of a P system, the set of reaction of a system can be written by means of rewriting rules, moreover, the communication of objects between adjacent volumes can be easily managed by using special kind of rewriting rules which send objects from one volume to another one.

A variant of membrane systems, called *dynamical probabilistic P systems* (DPPs), will be described in this chapter. Differently from the basic definition of P systems, in DPPs, probabilities are associated with the rules, and such values vary during the evolution of the system, according to a prescribed strategy. Thanks to the probability values, the rules are not applied in a maximal parallel way, but different level of parallelism can be used, thus allowing a qualitative description of the systems dynamics. DPPs represent the starting point of the development of a novel simulation strategy which allows the quantitative description of biological and chemical systems (that will be presented in Chapter 4). We present an application of DPPs for the modelling and simulation of *metapopulations*, also called multi-patch systems, which are ecological models used to analyse the behaviour of interacting populations, to the aim of determining how a fragmented habitat influences local and global population persistence.

3.1 Basic notions of membrane systems

Membrane systems, or *P systems*, were introduced in [153] as a class of parallel, distributed and nondeterministic computing devices, inspired by the architecture and the functioning of living cells. The fundamental features defined in membrane systems are the *membrane structure*, which identifies a (topologically) compartmentalised space, where *objects* can evolve according to specified *evolution rules*, which determine both the modification of the objects and their *communication* among membranes.

More precisely, the *membrane structure* consists of membrane-delimited compartments, hierarchically arranged and embedded inside a main compartment, whose delimiting membrane is called the *skin membrane*. A graphical representation of a membrane structures is given in Figure 3.1. The space of a compartment delimited by its membrane and the membranes of the other compartments (if any) placed immediately inside it, is called a *region*.

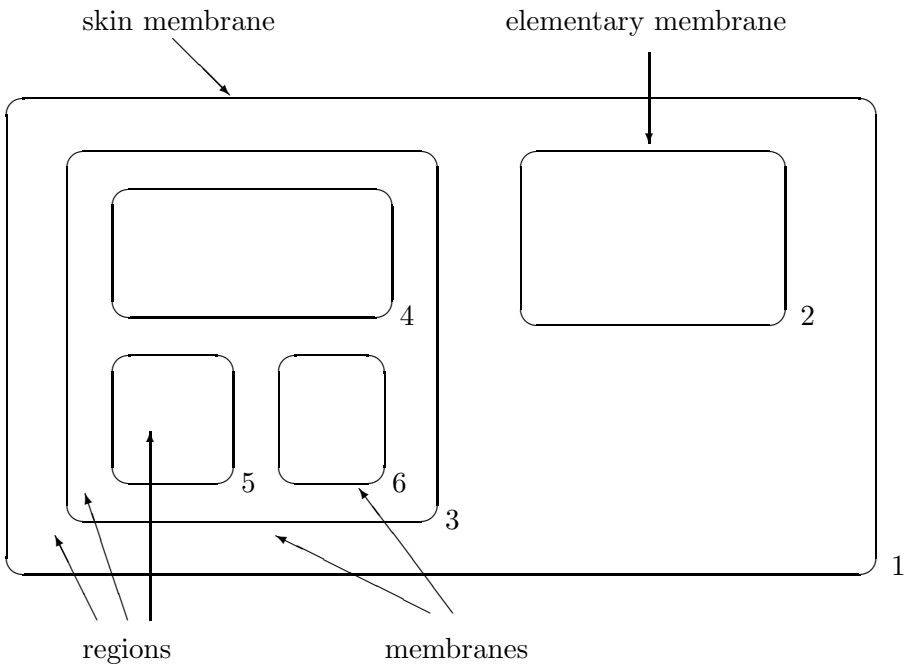


Figure 3.1: A membrane structure

With each region of the membrane structure we can associate a set (or a multiset) of *objects*, which can be symbols or strings over a given alphabet, and a set of *evolution rules*, which determine the transformation of objects, as well as their communication between adjacent compartments. So doing, different regions in the membrane structure can communicate with each

other by sending or receiving objects. Usually, the evolution rules are applied in a nondeterministic and maximally parallel way: all the objects which can evolve should evolve. Moreover, a strict partial order relation can be defined over the set of evolution rules present inside a compartment, indicating a priority relation among them. Due to the application of evolution rules, a membrane can also be dissolved: it disappears from the membrane structure, the set of rules inside that membrane is removed, the set of objects pass to the outer membrane (which contained the disappeared one). Hence, the evolution rules do not only modify the objects and their placing, but can also intervene in changing the membrane structure.

An "abstract" interpretation of a P system in biological terms can be given as follows. The membrane structure corresponds to the organisation of a living cell, of eukaryotic or prokaryotic type (see, e.g., [3]), where there exists an external membrane (the plasma membrane) and, possibly, some organelles inside the cell, each delimited by a membrane. In each organelle, as well as inside the cytoplasm, there exist different types of substances (ions, proteins, nucleic acids, etc.) which can be described by means of (multisets of) symbols or strings. Under specific conditions, these substances are modified by biochemical reactions and they can remain in the same organelle where they are placed, or selectively cross the membrane which delimits the organelle. These processes correspond to the way evolution rules are defined and work in P systems: they modify the formal objects (symbols or strings) and also indicate where the modified objects should be at the next step. The priority relations above rules can be interpreted as different "reactivities" of the biochemical reactions. Moreover, we can also assume that reactions not only modify chemicals, but also "consumes energy", thus when a rule with a higher priority is applied, no rule with a lower priority can be used too, because no more energy is available. Some biochemical reactions need the cooperation of catalytic substances, other may be so powerful to cause the membrane lysis. Both events have a natural correspondence in P systems: the use of catalyst objects, and in the dissolving action of rules. When an internal membrane is "broken" inside a cell, its content is left free in the cytoplasmic space, though the biochemical reactions previously occurring in that organelle might not take place in the cytoplasm, due to possibly different chemical and physical conditions (e.g., the pH value). Similarly, when a membrane is dissolved in a P system, its set of rules is removed, but its objects still remain in the membrane structure of the system.

A P system is a computing device obtained starting from an initial configuration (described by a fixed membrane structure containing a certain number of objects and rules) and letting the system evolve. A universal clock is assumed to exist: at each step, all rules from all regions are simultaneously applied to all objects which can be the subject of an evolution rule. When no further rule can be applied, the computation halts and we get the result in a prescribed way. On the contrary, if there is at least one rule that

can be applied forever, then the computation is said to be unsuccessful and no output is obtained.

In the following, the basic definitions of P systems, is given. For further details and for more information, examples and other variants, we refer to [153, 154, 156, 168].

Formally, the *membrane structure* can be represented by a string of correctly matching square parentheses, placed in a unique pair of matching parentheses. Each pair of matching parentheses corresponds to a membrane and, usually, membranes are univocally labelled with distinct numbers. For instance, the string $\mu = [_0\, [_1\,]_1\, [_2\, [_3\,]_3\, [_4\,]_4\,]_2\,]_0$ corresponds to a membrane structure consisting of 4 membranes placed at three hierarchical levels. Moreover, the same membrane structure can be also represented by the string $\mu' = [_0\, [_2\, [_4\,]_4\, [_3\,]_3\,]_2 [_1\,]_1\,]_0$, that is, any pair of matching parentheses at the same hierarchical level can be interchanged, together with their contents; this means that the order of pairs of parentheses is irrelevant, what matters is their respective relationship.

Each pair of matching parentheses [ ] appearing in a membrane structure μ is called a *membrane*. The external membrane of a membrane structure is called the *skin* membrane, while a membrane without any other membrane inside (appearing in a membrane structure in the form [ ]) is called an *elementary* membrane.

The number of membranes in μ is called the *degree* of μ and it is denoted by $deg(\mu)$. The *depth* of a membrane structure μ, $dep(\mu)$, is recurrently defined as:

1. if $\mu = [\,]$, then $dep(\mu) = 1$;
2. if $\mu = [\mu_1 \dots \mu_n]$, for some membrane structures $\mu_1, \dots, \mu_n$, then $dep(\mu) = 1 + \max\{dep(\mu_i) \mid 1 \leq i \leq n\}$.

A *region* is the closed space delimited by a membrane and by the membranes immediately inside it (if any). Note that a membrane structure of degree n contains exactly n regions, each one associated with a membrane and labelled with the same integer of that membrane. The whole space outside the skin membrane is called the *outer* region or *environment*.

Given a finite alphabet V, we can provide a membrane structure μ with some multisets $M_i : V \to \mathbb{N}$, one for each membrane $i \in \{0, \dots, n-1\}$.

The elements of a multiset are called *objects*, the multiset present in a region is called the *contents* of this region. The total number of objects in an elementary membrane i is called the *size* of i, denoted by $size(i)$. If a membrane j is placed inside a membrane i and they contribute to identify the same region (namely, the one delimited by the membrane i), then the objects inside the region associated with i are said to be *adjacent* to membrane j.

Finally, the objects occurring inside the regions of a P system can evolve, according to certain specified rules. An *evolution rule* over an alphabet V

is a pair (u, v), which can be written as $u \to v$, where u is a (non empty) string over the alphabet V and $v = v'$ or $v = v'\delta$, where v' is a string over $V \times (\{here, out\} \cup \{in_j \mid 0 \leq j \leq n-1\})$, and δ is a special symbol not in V. The length of the string u is called the *radius* of the rule $u \to v$. Note that the strings u, v are understood as representations of multisets over V, in a natural sense.

Given a membrane structure together with multisets and evolution rules, the formal definition [153] of the basic variant of P systems can be given as follows: a *transition P system* of degree $n, n \geq 1$, is a construction

$$\Pi = (V, T, C, \mu, M_0, \ldots, M_{n-1}, (R_0, \rho_0), \ldots, (R_{n-1}, \rho_{n-1}), i_o)$$

where:

- V is the alphabet of the system;
- $T \subseteq V$ is the terminal (or *output*) alphabet;
- $C \subseteq V, C \cap T = \emptyset$ is the alphabet of *catalysts*;
- μ is a membrane structure consisting of n membranes, which are injectively labelled by numbers in the set $\{0, \ldots, n-1\}$;
- $M_0, \ldots, M_{n-1}$ are multisets over V, representing the objects initially present in the regions $0, \ldots, n-1$ of the system;
- $R_0, \ldots, R_{n-1}$ are finite sets of *evolution rules* associated with the regions $0, \ldots, n-1$ of μ;
- $\rho_0, \ldots, \rho_{n-1}$ are strict partial order relations defined over $R_0, \ldots, R_{n-1}$ respectively, specifying a *priority relation* among the evolution rules;
- i_o is a number in the set $\{0, \ldots, n-1\} \cup \{\infty\}$, indicating the *output* region.

Some more remarks about the definition of a P system are to be given, before explaining its functioning as a computing device. Any multiset $M_0, \ldots, M_{n-1}$ can be empty, as well as any set of evolution rules $R_0, \ldots, R_{n-1}$, or any strict partial order relation $\rho_0, \ldots, \rho_{n-1}$.

If Π contains evolution rules whose radius is greater than 1, then it is said to be *cooperative*, otherwise it is a *non-cooperative* system. A particular class of cooperative systems is that of *catalytic* systems: whenever the radius of an evolution rule is strictly greater than 1, then such rule is of the form $ca \to cv$, where $c \in C, a \in V \setminus C, v \in (V \setminus C)^*$; moreover, no other rule contains any catalytic symbols. If $C = \emptyset$, then the system is *non-catalytic*.

According to the strict partial order relation $\rho_i, 0 \leq i \leq n-1$, a rule $r \in R_i$ is said to be *with priority* if there exists another rule $r' \in R_i$ such

that $r > r'$ or $r' > r$. If r is a rule with priority and there is no other rule r' such that $r' > r$, then r is a rule of *highest priority*. Note that, as ρ_i is a (strict) partial order relation, there can be more than one rule with highest priority. A rule r is said to be without priority if it does not appear in ρ_i.

The $(n+1)$-tuple $(\mu, M_0, \ldots, M_{n-1})$ constitutes the *initial configuration* of the system. In general, any sequence of the form $(\mu', M'_{i_1}, \ldots, M'_{i_k})$ constitutes a configuration of Π, where $M'_{i_1}, \ldots, M'_{i_k}$ are multisets over V and μ' is a membrane structure obtained by removing from the original membrane structure μ all membranes different from $i_1, \ldots, i_k$ (and not equal to the skin membrane), according to the dissolving action which will be explained later on. The labels $i_1, \ldots, i_k$ of the membranes in μ' are always numbers in the set $\{0, \ldots, n-1\}$, corresponding to the initial labels of the membranes in μ, that is, in successive configurations the initial labelling of the membranes (which have not been removed from μ) is never modified.

Given two configurations $C_1 = (\mu', M'_{i_1}, \ldots, M'_{i_k}), C_2 = (\mu'', M''_{j_1}, \ldots, M''_{j_l})$ of Π, we write $C_1 \Longrightarrow C_2$ and we say that there is a *transition* from C_1 to C_2 if we can pass from C_1 to C_2 by using the evolution rules in $R'_{i_1}, \ldots, R'_{i_k}$ using the following prescriptions.

All rules are simultaneously examined, in the decreasing order of their priority (if any), and objects are accordingly assigned to rules. A rule r can be applied only when there are enough copies of the objects whose evolution it describes, and which were not "consumed" by rules of higher priority. That is, objects are assigned to the rules which have the highest priority and, in the case that two or more rules of this type compete for the same objects, then the rule to be applied is nondeterministically chosen among them. Then, when no more rules with the highest priority can be applied (irrespectively of the objects they involve), the rules with a lower priority are considered. Again, if there are many rules with the same priority which compete for some objects, then the rule to be applied is nondeterministically chosen among them.

All the objects which *can* be the subject of evolution rules *must* be involved in the rule application. This use of the priority relations corresponds to the *strong interpretation*: if a rule with a higher priority is used, then no other rules with less priority can be applied, even if they do not compete for the same objects. Nonetheless, also a *weak interpretation* of the priority rules can be considered: a rule with less priority can be applied at the same time with the rule of higher priority if there are "free" objects which are not involved by the rule with the higher priority. The rules without priorities can be applied at any time, even together with rules with priorities.

The result of the application of the rule $(r : u \to v) \in R'_{i_s}$ is determined by v:

- if an object appears in v in the form $(a, here)$, then it remains in the same region i_s (the indication *here* will often be omitted);

- if an object appears in v in the form (a, out), then it exits the membrane i_s and becomes an element of the region immediately outside it. If the membrane i_s is the skin membrane, then the object leaves the system and it will never come back;
- if an object appears in v in the form (a, in_t) and i_t is a membrane immediately inside i_s, then the object is added to the multiset M_{i_t}. If i_t is not a membrane immediately inside i_s, then the application of the rule is not allowed;
- if the symbol δ appears in v, then membrane i_s is removed (we say it is *dissolved*) from μ' and, at the same time, the couple (R'_{i_s}, ρ_{i_s}) is erased, the multiset M'_{i_s} is added to the region immediately outside membrane i_s. The dissolving of the skin membrane is never allowed.

All these operations are performed in a maximal parallel mode: in one step, inside each region all applicable rules $u \to v$ are used over all occurrences of multisets u, and all regions are processed at the same time. Hence, the parallelism is maximal at both a global and a local level: the global level involves all membranes at the same time, the local level involves all evolution rules and all objects inside each region.

The functioning of the system as just described defines its nature as a *computing device*. Starting from the initial configuration and applying the rules in the way described above, we obtain a sequence of transition among configurations: such a sequence is called a *computation* with respect to Π. A computation is *successful* if and only if it halts, that is, no rule is further applicable to the objects appearing in any membrane of the system and, if an internal membrane i_o is specified as the output membrane, then it must appear in the halting configuration as an elementary membrane. If $i_o \in \{0, \ldots, n-1\}$, then the system is said to work in the *internal mode*, , and the result of a successful computation can be read in one of the following ways:

1. if we consider the multiset which is present in the output membrane, then the system is said to *generate multisets*;
2. if we consider the size of the output membrane, then the system is said to *generate numbers*;
3. if certain initial objects $a_{i_1}, \ldots, a_{i_k}$ are specified in advance, and we consider this occurrences $n_1, \ldots, n_k$ in the output membrane at the end of the computation, then the system is said to *generate relations*;
4. if a fixed elementary membrane is specified as the input membrane, the system is said to *recognise* a multiset (the initial contents of the input membrane), a number (the size of the input membrane) or a relation (the occurrences of certain symbols in the input membrane).

otherwise, the indication i_o for the output membrane can be omitted if $i_o = \infty$, in such a case the P system is said to be with *external output*, the objects which constitute the output of the system are collected outside the skin membrane, in the order they are ejected. Using these objects, a string is formed. When several objects are ejected at the same time, any permutation of them is considered. So doing, a *language* $L(\Pi)$ is generated by the system.

There exist several different variants of membrane systems; for instance, there are *tissue P systems* [121] which process symbols by means of multiset rewriting rules, within a net of cells (membranes). Each cell, having a finite state memory, processes multisets of symbol-impulses, and can send impulses to the neighbouring cells. This variant of membrane systems has been demonstrated to be rather powerful, indeed, it can simulate a Turing machine by using a small number of cells with a small number of states. There is another variant called *spiking neural P systems* [92] in which the idea of spiking neurons has been combined with the notions of membrane systems. Here, the computation depends upon the time when neurons fire or spike; indeed, the result of a computation is represented by the time elapsed before a specified neuron spikes. Both variants are mainly used as computing devices to study the complexity theory.

There exist other variants of membrane systems used to investigate biological systems; among others, there are *metabolic P systems* [114], which have been introduced for expressing the biological metabolism. Their evolution is given by a metabolic algorithm, a deterministic strategy where the classical viewpoint on metabolic dynamics in terms of ordinary differential equations, is replaced by generalised chemical principles. Another variant of membrane systems, called *conformon P systems* [66], has been introduce for the investigation of biological systems. Conformon P systems are based on simple and basic concepts inspired by a theoretical model of the living cell centred around conformons. A conformon is an object defined as a pair (name–value) and can be used to define modules that accomplish specific tasks. In [162], the class of dynamical probabilistic P Systems has been defined. The aim of this membrane systems variant is to provide a stochastic framework for the analysis and simulation of complex systems. This class of P systems represent a suitable framework which can be combined with the efficient stochastic simulation algorithms in order to define a novel tool for the study and analysis of biological systems (as described in Chapter 4).

3.2 Dynamical probabilistic P systems

In this section we recall the basic definitions of dynamical probabilistic P systems (DPPs), presented in [163, 162]. DPPs propose a new approach for the investigation and the application of P systems, which consists in inter-

preting them as stochastic tools for the description and the analysis of the dynamics of complex systems. In DPPs, probabilities are associated with the rules, and such values vary during the evolution of the system according to a prescribed strategy. The method for evaluating probabilities, the way the system works, the rule application method, and the corresponding software simulators are hereby explained. A complete and extensive description of DPPs, examples of simulated systems, and tools to analysed the properties of systems described by means of DPPs can be found in [163, 162, 161, 17, 16, 160].

Definition 3.2.1 *A* dynamical probabilistic P system *of degree n is a construct $\Pi = (V, O, \mu, M_0, \ldots, M_{n-1}, R_0, \ldots, R_{n-1}, E, I)$ where:*

- *V is the alphabet of the system, $O \subseteq V$ is the set of analysed symbols.*
- *μ is a membrane structure consisting of n membranes labelled with the numbers $0, \ldots, n-1$. The skin membrane is labelled with 0.*
- *M_i, $i = 0, \ldots, n-1$, is the initial multiset over V inside membrane i.*
- *R_i, $i = 0, \ldots, n-1$, is a finite set of evolution rules associated with membrane i. An evolution rule is of the form $r : u \xrightarrow{k} v$, where u is a multiset over V, v is a string over $V \times (\{here, out\} \cup \{in_j \mid 1 \leq j \leq n-1\})$ and $k \in \mathbb{R}^+$ is a constant associated with the rule.*
- *$E = \{V_E, M_E, R_E\}$ is called the environment, and consists of an alphabet $V_E \subseteq V$, a feeding multiset M_E over V_E, and a finite set of feeding rules R_E of the type $r : u \to (v, in_0)$, for u, v multisets over V_E.*
- *$I \subseteq \{0, \ldots, n-1\} \cup \{\infty\}$ is the set of labels of the analysed regions (the label ∞ corresponds to the environment).*

The alphabet O and the set I specify which symbols and regions are of peculiar importance in Π, namely those elements whose evolution will be actually considered during the analysis and the simulation of the system.

The *set of parameters* $\mathcal{P}$ of a dynamical probabilistic P system Π consists of:

1. the multiplicities of all symbols appearing in the multisets $M_0, \ldots, M_{n-1}$ initially present in μ, and of those appearing in the feeding multiset M_E;
2. the constants associated to all rules in $R_0, \ldots, R_{n-1}$.

The alphabets V, O, V_E, the membrane structure μ, the form of the rules in $R_0, \ldots, R_{n-1}, R_E$, and the set I of analysed regions do not belong to the

set of parameters of Π: these components are called the *main structure* of Π.

A *family* of DPPs is defined as $\mathcal{F} = \{(\Pi, \mathcal{P}_i) \mid \Pi \text{ is a DPP and } \mathcal{P}_i \text{ is the set of parameters of } \Pi, i \geq 1\}$, which represents a class of DPPs where all members have the same main structure, but the parameters can change from member to member. We assume that, given any two elements $(\Pi, \mathcal{P}_1), (\Pi, \mathcal{P}_2) \in \mathcal{F}$, it holds $\mathcal{P}_1 \neq \mathcal{P}_2$ for the choice of at least one value in $\mathcal{P}_1$ and $\mathcal{P}_2$. This is useful if one wants to analyse the same DPP with some different settings of initial conditions, such as different initial multisets and/or different rule constants (for instance, when not all of them are previously known and one needs to reproduce a known behaviour) and/or different feeding multisets. In other words, the family $\mathcal{F}$ describes a general model of the biological, chemical or ecological system of interest and, for any choice of the parameters, we can investigate the evolution of the corresponding fixed DPP.

Let us now describe the role played by stochasticity in DPPs. The probability associated with each rule in the set R_i, for all $i = 0, \ldots, n-1$, is a function of the rule constant and of the current multiset occurring in membrane i, and it is evaluated as follows. Let $V = \{a_1, \ldots, a_l\}$, M_i be the multiset inside membrane i and $r : u \xrightarrow{k} v$ a rule in R_i. Let also be $u = a_1^{\alpha_1} \ldots a_s^{\alpha_s}$, $alph(u) = \{a_1, \ldots, a_s\} \subseteq \{a_1, \ldots, a_l\}$ and $H = \{1, \ldots, s\}$. To obtain the actual normalized probability p_i of applying r with respect to all other rules that are applicable in membrane i at the same step, we first need to evaluate the (non-normalized) pseudo-probability $\widetilde{p_i}(r)$ of r, which depends on the constant associated with r and on the left-hand side of r, and is defined as:

$$\widetilde{p_i}(r) = \begin{cases} 0 & \text{if } M_i(a_h) < \alpha_h \text{ for some } h \in H, \\ k \cdot \prod_{h \in H} \dfrac{M_i(a_h)!}{\alpha_h!(M_i(a_h) - \alpha_h)!} & \text{if } M_i(a_h) \geq \alpha_h \text{ for all } h \in H. \end{cases} \tag{3.1}$$

Therefore, whenever the current multiset inside membrane i contains a sufficient number of each symbol appearing in the left-hand side of the rule (second case in Equation 3.1), then $\widetilde{p_i}(r)$ is dynamically computed according to the current multiset inside membrane i: for each symbol a_h appearing in u, we choose α_h copies of a_h among all its $M_i(a_h)$ copies currently available in the membrane, that is, we consider all possible distinct combinations of the symbols appearing in $alph(u)$. Thus, $\widetilde{p_i}(r)$ corresponds to the probability of having an interaction among the objects (placed on the left-hand side of the rule), which are considered undistinguishable.

A different probability distribution over rules could be used instead of Equation 3.1; the approach used here can be seen as an adaptation of the

Stochastic Simulation Algorithm, introduced in [74], to include and preserve the maximal parallelism of rule application. Nonetheless, to the aim of having the highest closeness to the biological system that one needs to model, other approaches could be investigated and compared as well. We also refer to [19], for a method with real-valued reaction maps, and to [32], where some stochastic approaches for biological modelling are reviewed and compared.

Definition 3.2.2 *An evolution rule is* mute*, if it is of the form* $u \to (u, here)$*, for some multiset* u *over* V*.*

Mute rules, which do not change the current multiset in any way, can be used as a trick to maintain the maximal parallelism at the level of rule application, but not at the level of object consumption. In other words, assume to have a membrane with two rules $a \to (b, out), a \to (c, here)$ and a multiset consisting of 10 copies of the symbol a. By using the maximal parallelism and the nondeterminism, in one step we would have α copies of c in the membrane, for some $\alpha \geq 0$, $10 - \alpha$ copies of b outside the membrane, and no more copies of a, since all occurrences had to be consumed. If the mute rule $a \to (a, here)$ is added inside this membrane, then in one step – and still using the maximal parallelism and the nondeterminism – we would obtain α' copies of c in the membrane, β copies of b outside the membrane, for some $\alpha', \beta \geq 0$, and $10 - \alpha' - \beta$ copies of a in the membrane. So doing, not all occurrences of a are consumed (to be more precise, they are actually consumed but also reproduced in the same form, in the same step), and thus the parallelism at the level of object consumption is no more maximal. Now, if we also associate stochastic constants to the rules, we can modulate the reduced parallelism by intervening on (the value of) the constant of the mute rule[1].

When adding mute rules to a membrane, we do not want to change the dynamical conditions which are determined inside that membrane by the constants associated to the other rules, and their respective mutual ratios. To this aim, we choose to apply the following strategy inside each membrane to compute the constant of *any* added mute rule, as well as the new constants of its corresponding rules. Let $S = \{r_1 : u \xrightarrow{k_1} v_1, r_2 : u \xrightarrow{k_2} v_2, \ldots, r_h : u \xrightarrow{k_h} v_h\} \subseteq R_j$ be a maximal subset of rules appearing inside a membrane j and having the same left-hand side, for some $u, v_1, \ldots, v_h$ multisets over V, with $v_i \neq u$ for all $i = 1, \ldots, h$, $h \geq 1$. Let us also define $K_S = \sum_{i=1}^{h} k_i$. For any maximal set S of this type (with different sets having different left-hand side multisets in the rules), we add the mute rule $r_{h+1} : u \xrightarrow{k_{h+1}} u$ to S, and we call $S' = S \cup \{r_{h+1}\}$. Since we want to preserve in S' the relative dynamics of the rules initially present in S, we have to associate

[1] We remark that the analogous concept of tuning the "activity" of rules is also used in the Metabolic Algorithm [19] with the name of *transparent rules*.

to the rules in S' new constants, whose values have to be chosen in order to satisfy the previous prescriptions about the dynamics and the mutual ratios of constants. So, let us denote by $k'_1, \ldots, k'_h$ the new constants of rules $r_1, \ldots, r_h$ in S', respectively, and let also be $K'_S = \sum_{i=1}^{h} k'_i$.

It is easy to verify that, by choosing

$$k'_i = \left(\frac{1}{1+\rho}\right) k_i \quad \text{for all } i = 1, \ldots, h, \tag{3.2}$$

and

$$k_{h+1} = \left(\frac{\rho}{1+\rho}\right) K_S, \tag{3.3}$$

we obtain that

1. the mute rule r_{h+1} will be applied with a certain proportionality with respect to the rules in S, that is, for some arbitrarily fixed $\rho \in [0, \infty)$ the following ratio holds:

$$\rho = \frac{k_{h+1}}{K'_S} \tag{3.4}$$

2. the addition of the mute rule to the set S does not modify the underlying dynamics, that is, it still holds

$$K'_S + k_{h+1} = K_S. \tag{3.5}$$

Once all necessary additional mute rules have been included in each membrane, the last step consists in the evaluation of the normalized probabilities for the rules. Hence, given the final set of rules $R'_i = \{r_1, \ldots, r_m\}$ appearing inside membrane i (possibly obtained from the initial set R_i by adding some mute rules), the normalized probability for any rule $r_j \in R'_i$ is

$$p_i(r_j) = \frac{\widetilde{p_i}(r_j)}{\sum_{j=1}^{m} \widetilde{p_i}(r_j)}\,. \tag{3.6}$$

A DPP works as follows. A fixed initial configuration of Π depends on the choice of $\mathcal{P}$, hence, it consists of the multisets initially present inside the membrane structure, the chosen rule constants and the feeding multiset, which is given as an input to the skin membrane from the environment at each step of the evolution (by applying the feeding rules). Different strategies in the feeding process can be used; for instance, in Section 3.3.1 we will show how to substitute the role played by the environment with some new feeding rules, of a stochastic type, which are directly defined inside the internal membranes.

At each step of the evolution, all applicable rules are simultaneously applied and all occurrences of the left-hand sides of the rules are consumed. We assume that the system evolves according to a universal clock (as in the basic definition of P systems), that is, all membranes and the application of all rules are synchronized. The applied rules are chosen according to the probability values dynamically assigned to them; clearly, the rules with the highest normalized probability values will be more frequently tossed. If some rules compete for objects and have the same probability value, then objects are nondeterministically assigned to them.

The simulation of the evolution of DPPs can be performed by means of a software written in C language, which has been developed for single processor PCs. In the simulations, the stochastic and parallel application of the rules is done by splitting each parallel step into several sequential sub-steps. By exploiting the fact that the probability distribution and the applicability of the rules are functions only of the left-hand side of the rules and their constants, each single parallel step is separated into three stages: in the first one, the probabilities of rules are evaluated; in the second one, a random number generator is used to choose the rule to be applied and to correspondingly consume the objects appearing in its left-hand side; in the third one, the multisets are updated using a stored trace of the rules previously tossed. Each stage starts only when the previous one has been applied to all the membranes in the DPP, and the same process is repeated for all evolution steps. A detailed description of the algorithm used for the simulation of DPPs, and of its complexity, can be found in [162].

The simulations reported in Section 3.3.2 have been run using an implementation of DPPs which exploit the MPI (Message Passing Interface) C libraries in order to: (*i*) distribute the computation over a cluster of processors to achieve scalability, (*ii*) have a direct mapping of the communications among the membranes, and (*iii*) speed up the computation.

Using this "parallel" version of the simulator for DPPs, each membrane evolves independently from the others for everything but the communication, such that it can reside on an independent MPI process. All the membranes of a DPP simultaneously evolve, and require to be synchronized at the end of each step to allow the communication process. Since the rules are applied in a maximal parallel way, in each membrane we can divide a step into three stages: (1) the computation of the probability distribution for the rules, (2) the assignment to the rules of all objects which can be modified by rules, (3) the communication and multisets updating. Further details on the MPI implementation of the DPPs simulation algorithm can be found in [32].

3.3 A DPP application: metapopulation systems

Metapopulations, also called multi-patch systems, are extensively investigated in Ecology to analyse the behaviour of interacting populations, to the aim of determining how a fragmented habitat influences local and global population persistence. A metapopulation consists of local populations, living in spatially separated habitat patches which can have different areas, quality or isolation, and a dispersal pool, which is the spatial place where individuals from a population spend some lifetime during the migration among patches. The dispersal of individuals is distance-dependent, it may reduce the local population growth and lead to an increase in extinction risk due also to environmental and demographical stochasticity, thus the persistence of populations is assumed to be balanced between local extinctions and establishment of new populations in empty patches [85]. In multi-patch systems, it is possible to distinguish between two principal classes of dynamics: the populations can locally interact inside a patch according to, e.g., the Lotka-Volterra model of preys and predators [145], while the dispersal of individuals among patches can have effects on the global behaviour of the whole system [93, 94, 191, 203].

The term "metapopulation" was introduced in [107], where a deterministic model for population dynamics of insect pests in agriculture has been developed. Lately, the topic has been largely applied to various populations species in natural or artificial/theoretical fragmented habitat landscapes. Among the various modelling methods used so far to investigate metapopulation dynamics, we here recall *stochastic patch occupancy models*, *population viability analysis* and *spatially explicit population models*.

Stochastic patch occupancy models (SPOMs) are based on the so-called presence/absence assumption, that is, they only model the occupancy state of habitat patches, without considering any local population dynamics, and neither the populations sizes are requested. Discrete-time SPOMs are homogeneous first-order Markov chains in which the occupancy pattern at any time $t + 1$ depends only on the pattern at time t; as a consequence, these models are well applied to organisms which have one generation per year, such as insects. Recently, a software for SPOMs, namely SPOMSIM, has also been developed for the automatic inclusion and analysis of several parameters and features (see [136]).

Population viability analysis (PVA) models are used in conservation assessment issues, well applied to single-species (e.g., rare or threatened species) populations analysis with respect to their risk – or expected time – of extinction, or their chance of recovery, thus measuring the viability requirements of the species and how likely it is for that species to persist along a certain future time. The reliability of these models and the parameter estimation are usually based on a large set of demographic, habitat and individual-based data, specific for the species under investigation. PVA

methods usually address aspects of management, such as impact of human activities, responses of species to reintroduction, captive breeding or nature reserves, and more. For additional information about this methodology and alternative ones for conservation assessment studies, see [2].

Spatially explicit population models (SEPMs) are individual-based or population-based models, that combine a simulator for the populations with a map of the landscape, such that landscape features can be explicitly incorporated into the model, to analyse how population dynamics is affected by any change in landscape (e.g., climate change, regional land-use, impact of fire or other global phenomena). Complex real-world landscape and the response of population to habitat changes can thus be investigated within these models. SEPMs require to know life history, demographic information and several habitat-specific data about the behaviours of dispersal and habitat-selection of the studied populations. Temporal or spatial variation of the landscape can also be incorporated, thus creating dynamical landscape where the distribution of suitable and unsuitable patches changes in time. Sensitivity analysis can be used within SEPMs for the estimation of parameters which are not available from ecological studies, e.g., for habitat selection or for dispersal behaviours, and to gain insights about the optimal and adaptive management strategies in varying contexts [40]. For further notions about SEPMs we refer to the review work by [52].

Whichever the modelling method is chosen for analysing a peculiar (emergent) behaviour of interacting populations in fragmented habitat, specific properties of the system can be, or even *have* to be, explicitly or implicitly considered, as the ones that we remark in the following (see [85, 136, 86] for further details).

Referring to the landscape, most models take care of the spatial structure, the local environmental quality, the patch area and connectivity (isolation), to grasp the effect of habitat fragmentation on species persistence, since high connectivity decreases local extinction risk (this is called the "rescue effect") and local conditions determines the growth and survival of patch populations. Moreover, dispersal and colonization are distance-dependent elements which give meaning to landscape structure in a spatial model. As for population interactions and dynamics, colonization can be related or not to cooperation of immigrating individuals (in the first case, it is called "Allee effect"). Within-patch dynamics is claimed to improve the understanding of natural systems; models which do not take care of it, thus only assuming whether a patch is occupied or not, usually consider local dynamics on a faster time scale with respect to the global dynamics, and also neglect the dependence of colonization and extinction rates on population sizes. Finally, regional stochasticity takes care of "bad" or "good" years over the local environmental quality. As we will explain later on, some of these issues will be explicitly considered in the next section for defining the membrane system model for metapopulations.

In what follows, we define a model of metapopulations with predator-prey dynamics by means of DPPs, where additional features are used in order to catch and better describe relevant properties of the modelled system. Namely, we will use a membrane structure of a novel type, where the spatial arrangement and the dimension of regions matter. With this choice we can give a proper description of a real (or artificial laboratory) landscape, and thus correctly analyse the dynamics of species interactions, via the definition of appropriate rule constants related to the fixed geometry. We then apply the generic model to some test case studies, and we investigate some emergent metapopulation behaviours, such as the effects of stochastic breeding, migration and colonization, isolation of patches, etc.

3.3.1 The modelling framework

Let μ be a membrane structure with degree $n+1$ and depth 2, with the skin membrane labelled with 0 and the internal membranes labelled with numbers $1, \ldots, n$. In the following, we will refer without distinction to the skin membrane or the *dispersal pool*, and to the internal membranes or regions or *patches*. For the modelling of a metapopulation, the membrane structure has to be intended as a set of spatially distributed regions, each one completed with a "size", which represent the area of the 2-dimensional patch, and with a fixed "distance" with respect to all other regions. Since the membrane structure is assumed to have here a precise spatial arrangement, we will not represent it as a string of well-matching square brackets where, as usual, any permutation of internal membranes would give rise to topologically equivalent structures (which is something that we want to avoid). Instead, to describe the set of internal membranes, we will use an undirected weighted graph with node attributes, thus outlining both the spatial distribution of patches and the relevant additional features associated to them: the *area* of a patch is needed to define the density of the populations living inside the patch (and thus to provide a better description of the dynamics), while the *distance* is needed to identify isolated patches, as well as to define the dispersal rates inside the pool. The distance among patches is not to be necessarily intended as a physical measure of how far a patch is from another one, but it can also represent a measure of how hard it is to move from one patch to another one due to, e.g., the geographical morphology of the landscape (presence of mountain chains, deep rivers or sea among the patches) or to other general local conditions (zones in the pool which cause high risks for survival). Hence, we will more generally call this distance the *cost*.

We also remark that, for metapopulation models, each region has to be considered as part of a 2-dimensional space, and not as a closed surface in a 3-dimensional space as it is usually implicit in P systems.

The membrane structure, the areas and the patch mutual costs are as-

sumed to be fixed in the model we propose here. The modification of these elements could be forced, and would indeed be necessary, in the analysis of landscapes whose main structure changes during time, such as patches disappearing or reducing their dimension (e.g., a fire burning a forest), patches increasing their dimension (e.g., new borders built around a protected park), or even patches in different positions in the pool (e.g., in laboratory experiments where patches are glass boxes that can be freely moved around).

Formally, to associate a spatial structure to the membrane structure, we define a weighted undirected graph $G = (N_\Delta, E, w)$ where:

- N_Δ is the set of nodes, or vertices, such that, to each node $x \in N_\Delta$, there is associated a value $a \in \Delta$, being Δ a set of attributes of some kind;
- $E \subseteq \{(x, y) \mid x, y \in N_\Delta\}$ is the set of (undirected) edges between nodes;
- $w : E \to \mathbb{R}^+$ is the weight function associating a cost to each edge.

In our case, the set of nodes N_Δ coincides with the set $\{m_1, \ldots, m_n\}$ of internal membranes, the attribute of a patch represents the *area* of the patch, the edges characterize which patch is directly reachable from any other patch (self-edges of the type (m_i, m_i) might exist as well), and the weight $w_{i,j}$ of an edge (m_i, m_j) represent a cost to measure the effort that individuals have to face when moving from patch m_i to m_j through the dispersal pool.

Note that, if a patch m_i is directly reachable from patch m_j, then the individuals exiting from patch m_i will (possibly) enter patch m_j after some time spent in the pool (and vice versa), thus we are not interested in finding whether there exists a path in the graph such that individuals exiting from m_i will enter and leave other patches in the graph path before finally arriving in m_j. Self-edges obviously represent the possible action of individuals exiting a patch and then entering it again. Moreover, we consider the case of undirected graphs, though also directed graphs with two possibly edges between any couple of patches can be used, and their relative costs can be different, thus better characterizing the landscape nature (e.g., the cost of climbing a mountain can be different from the cost of descending from it).

The *area* of a patch m_i, $i = 1, \ldots, n$, is a value $\sigma_i \in \mathbb{R}^+$. We will not define the area of the pool, we implicitly assume that it is large enough to contain all patches and to allow the dispersal of individuals, that is, $\sigma_0 \gg \sum_{i=1}^{n} \sigma_i$.

Definition 3.3.1 *Let V be the alphabet of population species. The* density *of patch m_i, $i = 0, 1, \ldots, n$, with respect to population species $X \in V$, is*

$$\delta_X(m_i) = \frac{|X|_{m_i}}{\sigma_i}, \tag{3.7}$$

where $|X|_{m_i}$ *denotes the number of individuals of species* X *inside patch* m_i*. The* total density *of patch* m_i *with respect to all population species is defined as*

$$\delta_V(m_i) = \frac{1}{\sigma_i} \sum_{X \in V} |X|_{m_i}. \tag{3.8}$$

Definition 3.3.2 *The* mean cost *of the graph* G *associated to the membrane structure* μ *is the value*

$$w_G = \frac{\sum_{i,j=1}^{n} w_{i,j}}{\sum_{k=1}^{n-1} n-k}. \tag{3.9}$$

Definition 3.3.3 *A patch* m_i*, for some* $i = 1, \ldots, n$*, is said to be* isolated *if* $w_{i,j} \gg w_G$*, for all* $j = 1, \ldots, n$*.*

In other words, a patch can be considered isolated if its distance from any other patch is very high, or if the cost to reach it (or to reach another patch when migrating from it) is much higher than the mean cost for reaching any other patch. As a consequence, the probability of the population individuals migrating from, or moving to, an isolated patch will be accordingly defined, in order to capture this characteristic of metapopulations.

Given all the necessary definitions, we can now introduce our DPP model Π for a generic metapopulation consisting on n patches, a dispersal pool, and two population species, one for preys and one for predators. We will assume that the populations dynamics inside each patch follows the Lotka-Volterra model (see [162] for a previous description with DPPs) and that, without loosing in generality, the sustenance resources are identical for all species. Hence, let

$$\Pi = (V, O, \mu, I, G, M_0, \ldots, M_n, R_0, \ldots, R_n)$$

be such that

- $V = \{A, A', X, Y, Y_1, \ldots, Y_n\}$ consists of the symbol A for sustenance resources, X for the species of preys, Y for the species of predators. A' is used to simulate the stochastic feeding of resources, while $Y_1, \ldots, Y_n$ are used for denoting, via the subscripts $1, \ldots, n$, which is the originating patch of the predators occurring in the pool, as it will be clear below.
- $O = \{X, Y\}$ is the set of analysed symbols.
- μ is a membrane structure with n elementary membranes, each corresponding to a patch in the metapopulation system. The skin membrane plays the role of the dispersal pool.

- $I = \{1, \ldots, n\}$ is the set of analysed regions.
- G is the graph associated to the internal membranes, where we specify the set of area attributes for patches, $\Delta = \{\sigma_1, \ldots, \sigma_n\}$, and the set of costs associated to the edges, $\Omega = \{w_{i,j} \mid (m_i, m_j) \in E, \text{ for all } i, j = 1, \ldots, n\}$;
- $M_0 = \emptyset$ and $M_i = \{A' X^{p_i} Y^{q_i}\}$, for some $p_i, q_i \in \mathbb{N}$, $i = 1, \ldots, n$, are the multisets initially present inside the regions.
- For each $i = 1, \ldots, n$, R_i contains the following rules:

$$
\begin{aligned}
r^i_{(\text{feeding})} &: A' \xrightarrow{k^i_f} (A' A^{\alpha_i}, here),\ \alpha_i \in [N_{i_1}, N_{i_2}],\ N_{i_1}, N_{i_2} \in \mathbb{N},\\
r^i_{(X\text{growth})} &: AX \xrightarrow{k^i_{Xg}} (XX, here),\\
r^i_{(Y\text{growth})} &: XY \xrightarrow{k^i_{Yg}} (YY, here),\\
r^i_{(X\text{death})} &: X \xrightarrow{k^i_{Xd}} (\lambda, here),\\
r^i_{(Y\text{death})} &: Y \xrightarrow{k^i_{Yd}} (\lambda, here),\\
r^i_{(\text{dispersal})} &: YY \xrightarrow{k^i_d} (Y, here)(Y_i, out),
\end{aligned}
$$

and the corresponding mute rules:

$$
\begin{aligned}
r^i_1 &: AX \xrightarrow{k^i_1} (AX, here),\ \text{for some } \rho^i_1 \in [0, \infty),\\
r^i_2 &: XY \xrightarrow{k^i_2} (XY, here),\ \text{for some } \rho^i_2 \in [0, \infty),\\
r^i_3 &: X \xrightarrow{k^i_3} (X, here),\ \text{for some } \rho^i_3 \in [0, \infty),\\
r^i_4 &: Y \xrightarrow{k^i_4} (Y, here),\ \text{for some } \rho^i_4 \in [0, \infty),\\
r^i_5 &: YY \xrightarrow{k^i_5} (YY, here),\ \text{for some } \rho^i_5 \in [0, \infty),
\end{aligned}
$$

with $k^i_f, k^i_{Xg}, k^i_{Yg}, k^i_{Xd}, k^i_{Yd}, k^i_d, k^i_1, \ldots, k^i_5 \in \mathbf{R}^+$, $i = 1, \ldots, n$. Rule $r^i_{(\text{feeding})}$ describes the stochastic feeding of sustenance resources, that is, at each time step α_i copies of symbol A are created in region i, for some α_i randomly chosen in a previously fixed range $[N_{i_1}, N_{i_2}]$. These available resources then allow the growth of preys by means of rule $r^i_{(X\text{growth})}$. Rule $r^i_{(Y\text{growth})}$ governs the direct interactions among preys and predators, and the consequent growth of predators, while rule $r^i_{(\text{dispersal})}$ describes the action of predators migrating from patch m_i into the dispersal pool. The use of the multiset YY in the left-hand side of this rule allows to account for the predator density in the patch

during the process of migration, as will be better explained below. Note that, when a predator Y exits patch m_i, it arrives in the pool denoted as Y_i: the subscript is needed in order to distinguish, inside the pool, among predators migrating from different patches, who then colonize (see rules $R^0_{(\text{colonization})}$ in the set R_0) other patches according to the mutual distance of its originating patch with respect to the others placed inside μ. Rules $r^i_{(X\text{death})}, r^i_{(Y\text{death})}$ describe the death of individuals for causes which are not dependent to direct inter-species interactions. Finally, $r^i_1, \ldots, r^i_5$ are the mute rules allowing the non-maximal consumption of individuals, as explained in Section 3.2. Note that we do not add the mute rule corresponding to $r^i_{(\text{feeding})}$, since the net effect of this rule (creation of a certain number of A's) is already non "maximally parallel", by construction.

- R_0 contains the subsets of rules

$$\begin{aligned} R^0_{(\text{colonization})} &: \{Y_i \xrightarrow{k^0_{c,i,j}} (Y, in_j) \mid (m_i, m_j) \in E, i, j = 1, \ldots, n\}, \\ R^0_{(\text{death})} &: \{Y_i \xrightarrow{k^0_{d,i}} (\lambda, here) \mid i = 1, \ldots, n\}, \\ R^0_{(\text{wandering})} &: \{Y_i \xrightarrow{k^0_{w,i}} (Y_i, here) \mid i = 1, \ldots, n\}, \end{aligned}$$

with $k^0_{c,i,j}, k^0_{d,i}, k^0_{w,i} \in \mathbf{R}^+$. The rules in the subset $R^0_{(\text{colonization})}$ describe the process of colonization of patches m_j by predators Y_i which migrated from patch m_i. Note that, when entering a patch, a predator Y_i looses its subscript and is again denoted by Y, since now it is no more necessary to know from which patch it was coming from (remember that only one species of predators is considered, so the symbols $Y_1, \ldots, Y_n$ represent only a trick and do not correspond to different predator species). Rules in $R^0_{(\text{wandering})}$ describe the life spent by predators inside the dispersal pool during the migration, and might be seen (but we will not consider them as such) as the respective mute rules of the rules in $R^0_{(\text{death})}$, which describe the death of predators during the migration.

With respect to the definition of a DPP as given in Section 3.2, one can note that there are two main differences: here we do not consider the environment, whose role is somehow played by the dispersal pool, and the feeding rules that were defined inside the environment are here substituted with "resource creating" rules, which are active inside the patches. Moreover, the new feeding rules are of a stochastic type, thus we can more precisely describe different real situations where the amount of resources changes in time (due to, e.g., different weather conditions or environmental quality).

We describe in the following how to evaluate the probabilities of some rules, in order to let the dynamics of the system depend upon the geometrical structure of the patches, the initial population multisets and the patch densities (thus implicitly considering also the patch areas). Namely, for these rules, the expression given in Equation 3.1 will have some additional term.

Let us consider first the rules $R^0_{(\text{colonization})}$ in the pool, governing the colonization of patches. The application of these rules has to depend upon the cost between the patch from which the individual migrated and the patch that the individual will colonize, meaning that the higher the cost between two patches, the lower the probability for reaching one of them once the other has been left. Thus, let $r : Y_i \xrightarrow{k^0_{c,i,j}} (Y, in_j)$ be a rule in $R^0_{(\text{colonization})}$, and let $w_{i,j}$ be the cost associated to the edge $(m_i, m_j) \in E$, then

$$p(r) = \frac{1}{w_{i,j}} \cdot \widetilde{p}(r) \tag{3.10}$$

where $\widetilde{p}(r)$, given by Equation 3.1, is the usual term that takes care of the rule constant $k^0_{c,i,j}$ and of the combinatorial part, analogously to the Stochastic Simulation Algorithm [74].

Then, the rules inside the patches governing the growth and dispersal have to depend on the local populations densities. Hence, the probabilities of these rules will be evaluated as follows:

- for $r^i_{(X\text{growth})} : AX \xrightarrow{k^i_{Xg}} (XX, here)$ we use

$$p(r^i_{(X\text{growth})}) = \frac{1}{\sigma_i} k^i_{Xg} |A|_{m_i} |X|_{m_i} \tag{3.11}$$

  meaning that the higher the number of preys inside the patch, the higher the competition for food resources among them, hence the lower their growth;

- for $r^i_{(Y\text{growth})} : XY \xrightarrow{k^i_{Yg}} (YY, here)$ we use

$$p(r^i_{(Y\text{growth})}) = \frac{1}{\sigma_i} k^i_{Yg} |X|_{m_i} |Y|_{m_i} \tag{3.12}$$

  meaning that the higher the number of preys, the higher the growth of predators but, at the same time, if too many predators are present inside the patch, then their growth is reduced;

- for $r^i_{(\text{dispersal})} : YY \xrightarrow{k^i_d} (Y, here)(Y_i, out)$ we use

$$p(r^i_{(\text{dispersal})}) = \frac{1}{\sigma_i} k^i_d \frac{|Y|_{m_i}(|Y|_{m_i} - 1)}{2} \tag{3.13}$$

meaning that the higher the number of predators, the higher the probability that they will leave the patch in search of new colonizable patches for better survival chances.

Finally, the constant of each mute rule is evaluated according to the strategy described in Section 3.2, by specifying its chosen proportionality value $\rho_j^i \in [0, \infty)$, for all $i = 1, \ldots, n, j = 1, \ldots, 5$.

The set of parameters of Π is $\mathcal{P} = \{p_i, q_i, N_{i_1}, N_{i_2}\} \cup \{k_f^i, k_{Xg}^i, k_{Yg}^i, k_{Xd}^i, k_{Yd}^i, k_d^i, k_1^i, \ldots, k_5^i, k_{c,i,j}^0, k_{d,i}^0, k_{w,i}^0\} \cup \{\rho_1^i, \ldots, \rho_5^i\}$, $i, j = 1, \ldots, n$. A family of DPPs for metapopulations consists of a set of DPPs of type Π having the same underlying graph-membrane structure, but different choices of initial multisets, resource feeding range and rule constants.

3.3.2 Simulations and results

Several different simulations have been run to the aim of outlining the role of the various features of the DPP model for metapopulations [16, 17]. We stress the fact that we are not considering real data about a fragmented habitat or interpopulation dynamics of a specific ecological metapopulation, but we would rather show the effectiveness of our proposed framework in the investigation of the relevant characteristic of metapopulations. In this section we firstly describe how each feature of the model independently influences the dynamics of the system, then some interactions between these features will be shown.

In all the following examples we consider the spatially arranged membrane structure given by $\mu = [_0\, [_1\,]_1\, [_2\,]_2\, [_3\,]_3\, [_4\,]_4\,]_0$ and by a complete underlying graph without self-edges. The initial multisets, unless otherwise specified, correspond to $p_i = q_i = 1000$ for all $i = 1, \ldots, 4$.

In the first group of simulations, the investigation of the effect of each characteristic is organized in such a way that – among the four patches, when possible – patch 1 is used as the reference one and its behaviour is left unaltered with respect to the Lotka-Volterra model, while the other three patches show a range of application of the feature in exam.

We begin focusing on the role of the patch area; in order to isolate this effect, we choose to apply a constant feeding ($\alpha_i = 200$ for all $i = 1, \ldots, 4$) inside each patch and to deny the dispersal of the predators, such that each patch evolves independently (that is, all constants of rules in R_0 are null). The area of patch 1 is $\sigma_1 = 1$, while the other three values are $\sigma_2 = 0.35$, $\sigma_3 = 1.5$, $\sigma_4 = 2.5$. The chosen rule constants inside the patches are $\mathcal{C}_1 = \{k_{Xg}^i = 0.1, k_{Yg}^i = 0.01, k_{Xd}^i = 0, k_{Yd}^i = 10, k_d^i = 0 \mid i = 1, \ldots, 4\}$, $\rho_j^i = 0$, for all $i = 1, \ldots, 4, j = 1, \ldots, 5$ (that is, mute rules are not applied). In Figure 3.2 (left), the dynamics in the phase space of the relative model is presented. Clearly, as the area of patches increases from patch 2 to patch 4, also the probability of extinction of the populations increases (Figure

3.2, right). Moreover, we have noticed that the dynamics of the smallest patch, namely patch 2, does not obeys to a pure Lotka-Volterra behaviour, but only a minimal stochastic variation around the value of 400 individuals occurs (graphic not shown). It can also be seen that patch 4 undergoes the extinction of predators, which allows the consequent uncontrolled increase of preys (Figure 3.2, right).

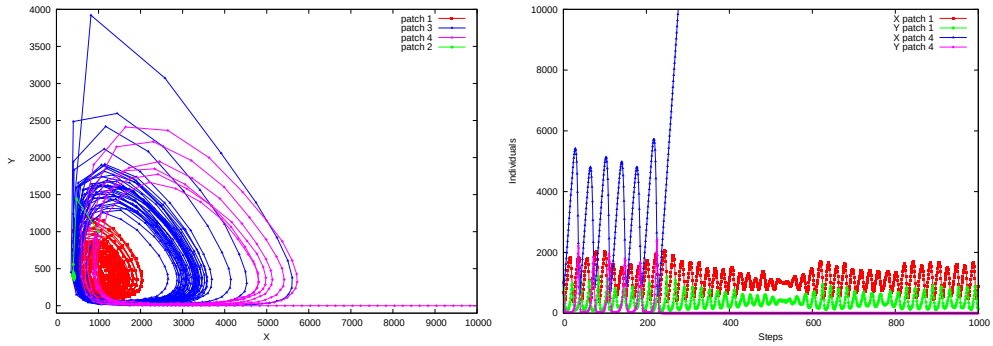


Figure 3.2: The role of patch areas.

In Figure 3.3 we start investigating the role of mute rules: in a structure with equal areas and where no dispersal occurs, inside patches 2, 3 and 4 we define different ratios for mute rules, namely $\rho_i^2 = 1/3, \rho_i^3 = 1, \rho_i^4 = 3$, for all $i = 1, \ldots, 5$. Patch 1 is still used as the reference one, so mute rules are not applied there. The initial rule constants of all patches are equal to those defined in the set $\mathcal{C}_1$, the actual constants of initial and mute rules used during simulations have then been evaluated as described in Section 3.2. What turns out in this case is that the highest the chance of executing mute rules is (patch 4), the lowest the variation in the numbers of preys and predators is. This is easily explained by the fact that the application of mute rules diminishes the number of modified objects inside membranes.

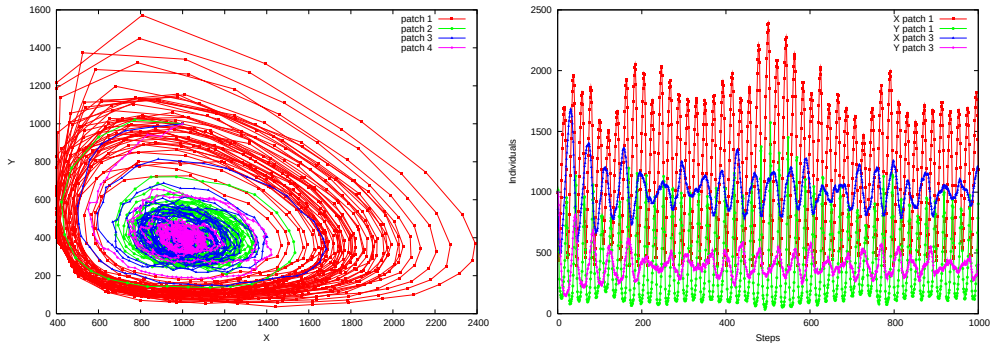


Figure 3.3: The role of mute rules.

The role of different stochastic feeding is analysed for patches having

all area values equal to 1 and where the dispersal of predators is still not allowed (hence no rules inside the pool are used). Once more, the constants of rules inside the patches are those in the set $\mathcal{C}_1$, mute rules are not active and $\alpha_1 \in [0, 200]$, $\alpha_2 \in [50, 200]$, $\alpha_3 \in [100, 200]$, $\alpha_4 \in [150, 250]$. It is possible to see that inside patch 1, which has the lowest feeding range, the extinction of predators is immediate (red line in Figure 3.4, top), in patch 2 the extinction of predators happens approximately at step 600 (Figure 3.4, bottom left), while in patches 3 (Figure 3.4, bottom right) and 4 (data not shown, but similar to patch 3), which have the highest feeding ranges, there still exists persistence of oscillating behaviour among preys and predators.

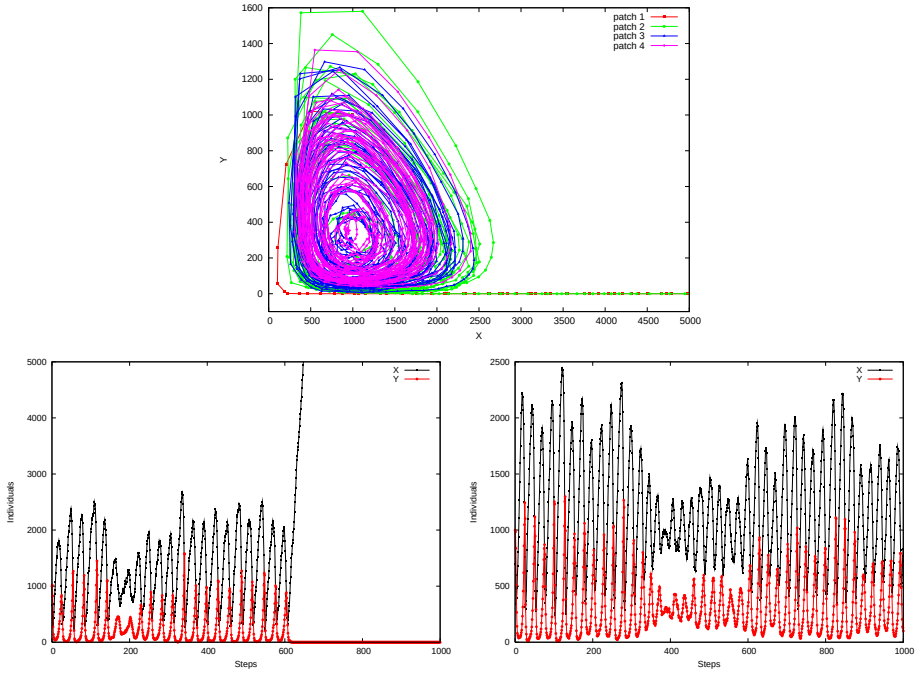


Figure 3.4: The role of stochastic feeding.

Then, in Figure 3.5 we focus on the role of edge costs, which directly affect the migration of predators among (connected) patches. Therefore, now it is not possible to have a reference patch since any patch is no more independent from the others. All patches have the same area, a constant identical feeding ($\sigma_i = 1$ and $\alpha_i = 200$, for all $i = 1, \ldots, 4$), and cost equal to 1 for all edges but $\omega_{2,3} = 0.01$. The rule constants are the same as those used in the first simulation, that is $\mathcal{C}_1$, but $k_{\mathrm{d}}^i = 0.01$ for all $i = 1, \ldots, 4$. In this case, it is possible to see that the dynamics inside patch 3, which is more easily reachable from patch 2, is different from the dynamics of the other patches: a larger number of predators enter this patch, thus the rate of growth of preys is correspondingly reduced.

In what follows we present some examples where the interaction between

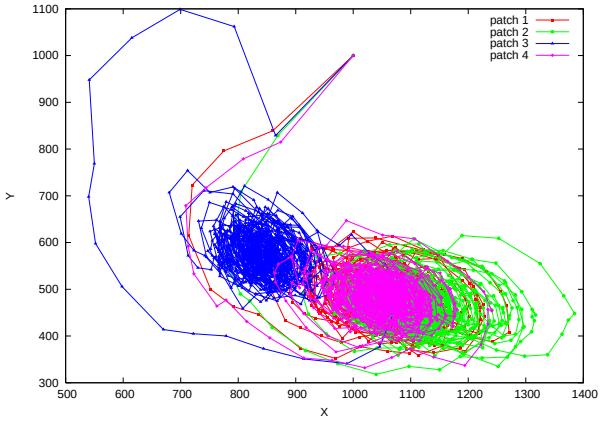


Figure 3.5: The role of costs.

different features of the model is analysed.

We consider the case of an isolated patch (patch 4), modelled by having an high reachability/escape cost ($\omega_{4,j} = 1000, j = 1,2,3$) and a slightly lower constant for predators dispersal ($k_d^4 = 0.001$) than the value defined in the other patches ($k_d^i = 0.01$, $i = 1,2,3$). All other rule constants inside the patches are equal to those in set $\mathcal{C}_1$, while the constants in R_0 are $\mathcal{C}_2 = \{k_{d,i}^0 = k_{w,i}^0 = 0, k_{c,i,j}^0 = 1 \mid i,j = 1,\ldots,4\}$. Patch 4 has also a small area ($\sigma_4 = 0.4$, while $\sigma_1 = \sigma_3 = 1.5$, $\sigma_2 = 1$) and no predators are initially present inside it. The costs among all other patches are $\omega_{1,2} = 100$, $\omega_{1,3} = \omega_{2,3} = 1$, mute rules are not applied and the same range of stochastic feeding ($\alpha_i = [150, 250]$, $i = 1,\ldots,4$) is defined inside all patches. We can see from simulations in Figure 3.6 (left) that inside the isolated patch there is an initial explosion in preys growth but, as soon as predators manage to reach this patch, then the oscillations in the numbers of preys and predators remains inside a reduced range (around 200-800 individuals for each species), due to the small patch area. On the contrary, the dynamics of the other patches are similar but balanced by the different areas (Figure 3.6, right).

We extend the previous simulation by considering a landscape with an isolated patch and another patch (we call it the "desert") where the natural death of preys can happen. The additional feature is obtained by setting $k_{Xd}^3 = 0.01$, while $k_{Xd}^i = 0$ for $i = 1,2,4$. Again, no mute rules are applied. In Figure 3.7 (left) we see that the behaviour of the isolated patch 4 is similar to the previous one, while the dynamics of the desert, patch 3, is very different from the other ones: even if its area is larger than the others, no real oscillating behaviour emerges. The rapid variation in the number of preys and predators that can be seen in Figure 3.7 (right) mainly correspond to stochastic noise.

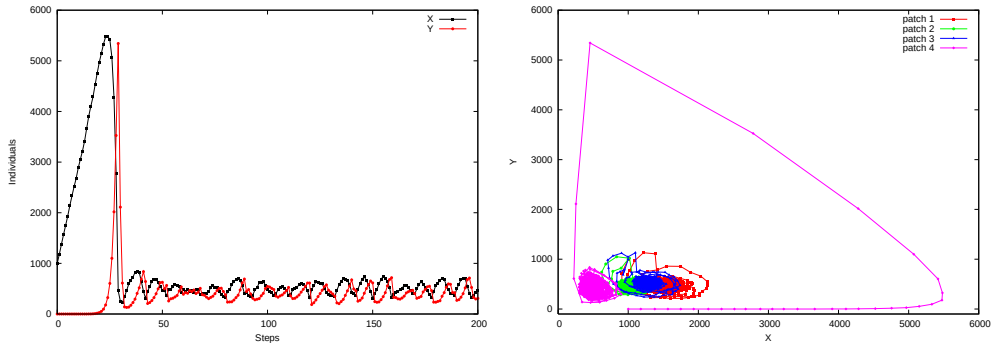


Figure 3.6: An isolated patch.

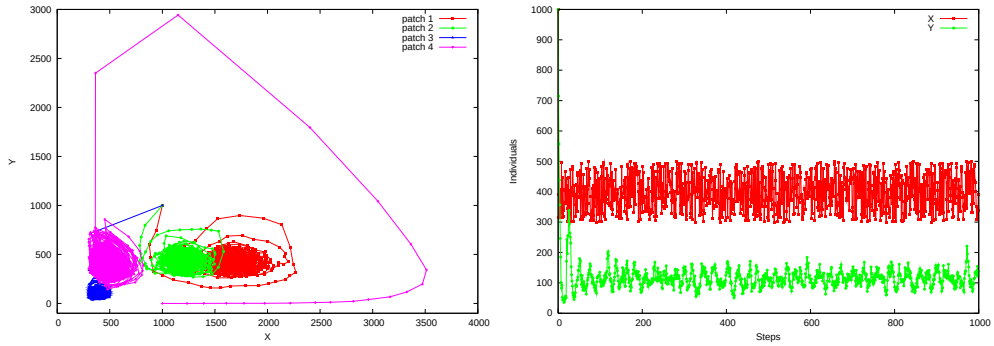


Figure 3.7: An isolated patch and a desert.

We now pass to analyse the behaviour of a system where patches have different areas ($\sigma_1 = 0.5, \sigma_2 = 0.7, \sigma_3 = 1.4, \sigma_4 = 2.1$), a stochastic feeding in the range $[150, 250]$ is defined inside all patches, all edges have the same cost equal to 1 and the ratios of mute rules are $\rho_j^i = 1/3$, for $i = 1, \ldots, 4, j = 1, 2, 4, 5$, $\rho_3^i = 9$, for $i = 1, \ldots, 4$. The initial constants in the patches are the same as in set $\mathcal{C}_1$. Here we can see (Figure 3.8, left and right) how the dynamics of patches 3 and 4 vary with respect to patches 1 and 2 due both to their bigger areas and to the rules that govern the life/death of preys.

Analysing the role of different costs ($\omega_{1,2} = 10, \omega_{1,3} = \omega_{1,4} = 50, \omega_{2,3} = 1, \omega_{2,4} = \omega_{3,4} = 30$) and different areas ($\sigma_1 = 0.5, \sigma_2 = 0.7, \sigma_3 = 1.4, \sigma_4 = 2.1$), while the ratios of mute rules are fixed at a same value inside all membranes (as in the previous simulation), we can see from Figure 3.9 (left) that the dynamics of patches 3,4 are different from patches 1,2 since the former have higher reachability costs and bigger areas. In Figure 3.9 (right) the species behaviours of patches 1 and 4 are directly compared.

Finally, in Figure 3.10 we present the role of poor stochastic feeding inside patch 4 ($\alpha_4 \in [50, 100]$, with respect to $\alpha_i \in [150, 250], i = 1, 2, 3$), where all areas and costs are equally set to 1, and mute rules are not applied.

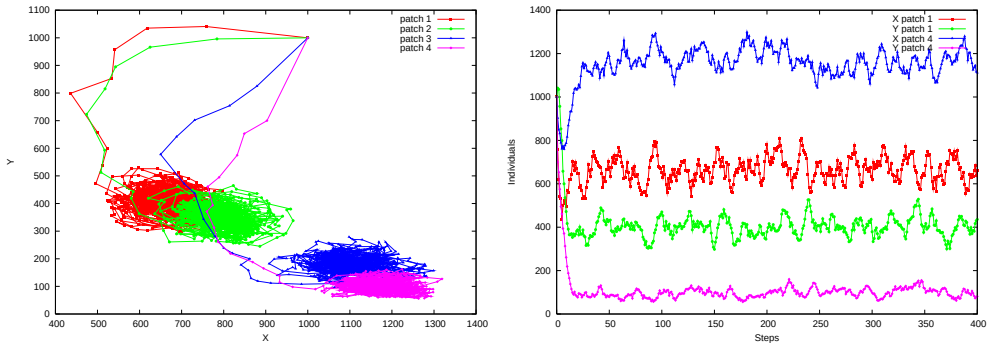


Figure 3.8: Mixed features: different areas, stochastic feeding, mute rules, equal costs.

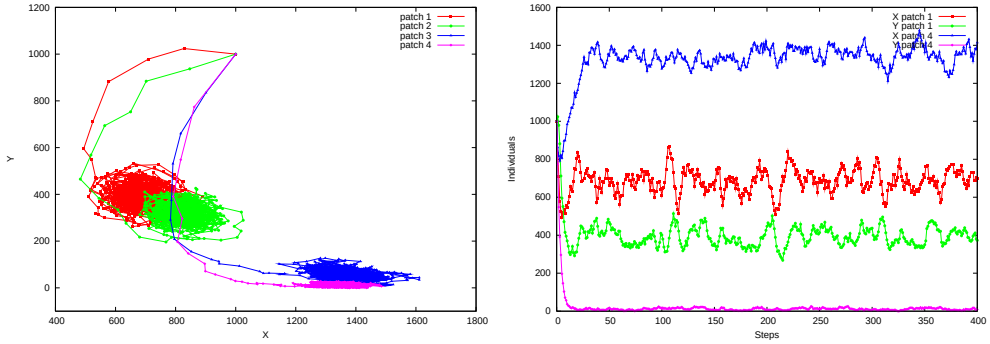


Figure 3.9: Mixed features: different areas and costs, stochastic feeding, mute rules.

We can thus see that the lower feeding causes a decrease in the number of preys and, above all, in the number of predators, with respect to all other patches.

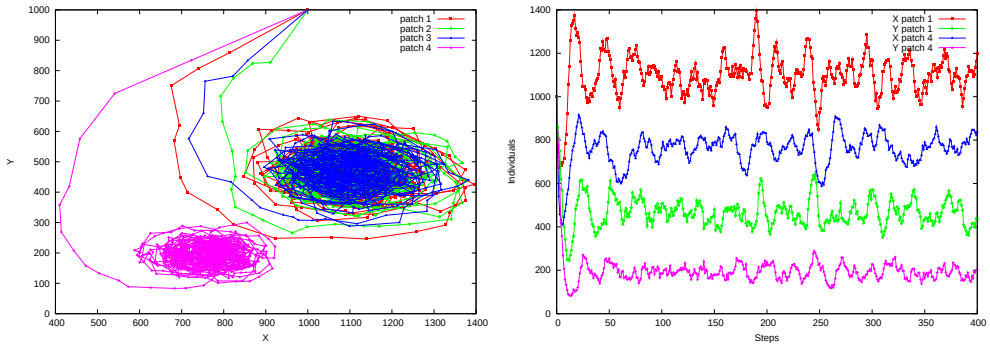


Figure 3.10: Reduced stochastic feeding.

3.3.3 Discussion

In Section 3.3.1 a metapopulation system with 2-species predator-prey dynamics has been considered. The proposed DPP model can be easily extended to cases where interaction of more species occur, as well as to represent other metapopulation dynamics. For instance, we have only considered the case of predator's dispersal, though also prey's dispersal can be easily added in a more general framework. Moreover, the type of membrane structure with the additional features we introduced for modelling metapopulations, which allows to define regions with a specific area and disposition in space, might also be useful for the analysis of epidemiologic systems.

In the work presented in [16], the influence of different resource feeding strategies on the metapopulations dynamics has been analysed. In particular, the comparison between systems where increasing, decreasing, stationary of purely feeding stochastic phases defined inside the patches, has been presented. The obtained results have shown how the seasonal variance of the resources feeding can transform the basic (stochastic) Lotka–Volterra dynamics inside each patch into a more complex dynamics, where different phases of a feeding cycle can be identified.

Some other relevant characteristics of ecological metapopulations, which can be included in DPP model, are: (*a*) the classification of species individuals as strong/weak colonizer, or strong/weak rescuer [171]; (*b*) the use of clusters of patches [86]; (*c*) the role of "synchrony" on metapopulation extinction (where synchrony means that the population dimensions in close patches tend to have similar fluctuations), in such a way that the presence of synchrony causes local extinction to determine also global extinction, while in the absence of synchrony the metapopulation has higher chances of persistence [109]; (*d*) the roles of "predator pursuit" and "prey evasion" on the (predators and preys) migration and colonization, both depending on the densities of patches [109].

Moreover, the application of this modelling framework to real available data might contribute to a further investigation of metapopulations, allowing a direct comparison between this stochastic discrete approach and the other modelling frameworks in Ecology, such as those reviewed in Section 3.3. Here we just outline that, for example, one of the limits of SEPMs is that, within this model, quantitative data such as the size of a population inside patches cannot be determined, but only qualitative analysis can be done. The P systems-based model proposed here allows to know, instead, the exact number of individuals of any species in any region (patches or pool) of the system – a very relevant aspect when dealing with populations with a high risk of extinction or severely damaged. On the other hand, with SEPMs one can account for the real passing of time and thus investigating the population dynamics over time ranges from days to years; in the current membrane system model we developed for metapopulation, it is not possible

to include such feature yet, though accounting for real evolution time is a hot topic in P systems and already a work in progress [30, 20].

Chapter 4

τ–DPP

In this chapter, a novel algorithm for the simulation of chemical and biological systems composed by many volumes will be described. This approach, called τ–DPP, has been first presented in [30]; it combines some features of membrane systems, in particular of the DPP variant (see Section 3.2) with the simulation strategy used in the tau-leaping algorithm (see Section 1.3). τ-DPP exploits a particular kind of reactions, called *communication rules*, to move objects between volumes, rather than considering diffusive processes (as, for instance, in the NSM [57] and in the Bτ-SSSA [117]). Therefore, both "classic" reactions and communication rules will be selected using the same strategy.

τ-DPP has been introduced to the aim of extending the single-volume algorithm of tau-leaping [26], in order to handle multi-volume systems where distinct volumes are arranged according to a specified hierarchy, and with the additional assumption that the topological structure and the size of the volumes do not change during the system evolution. As already seen in Chapter 3, this condition is well satisfied by the membrane structure of a P system, which is also the underlying spatial arrangement used within τ-DPP. Indeed, τ-DPP presents a close correspondence to DPP, but the first allows a *quantitative* description of the system's dynamics, while in the second the obtained behaviour is only *qualitative*.

In order to correctly describe the behaviour of a system, τ-DPP runs in parallel inside each volume. A modified version of the tau-leaping procedure presented in [26] is exploited for the computation of the length of the step τ. In this novel version of the simulation algorithm, the least value for the time increment, among those computed inside each volume, is used to sample the number of reactions to execute (as in the original tau-leaping algorithm). Thanks to this "common" time increment, shared by all volumes, the simulation is synchronized at each step, allowing the correct passage of the molecules involved in communication rules.

The computational cost of the τ-DPP algorithm is the same as the tau-

leaping strategy, for a single volume. Clearly, the time needed for a simulation increases with the number of volumes composing the system. Nevertheless, it is very easy to distribute the computation over a parallel architecture, where volumes are simulated on different nodes which communicate at the end of each iterative step to synchronize their dynamics.

In the second part of the chapter, a recently presented variant of τ-DPP will be described [29]. Using this approach, the size of volumes and objects is considered, in order to better describe systems where the "space" play an important role in the dynamics (such as, for instance, reaction–diffusion systems).

We conclude with some discussion about the modelling power of τ-DPP, and future developments for this approach, which consist, among others, in the development of different strategies for the selection of the reactions, and the inclusion of the "membrane potential" in the computation of the propensity functions of particular cellular processes.

4.1 Tau-leaping procedure in DPPs

The introduction of τ-DPP [30] for the simulation of chemical and biological systems has been motivated by two main problems. The first consists in the fact that the tau-leaping method [26], which is one of the finest stochastic simulation algorithms, is only applicable to well stirred chemical reaction systems contained inside a *single* fixed volume. The second problem concerns DPPs, which can only allow *qualitative* simulations of a system's dynamics; moreover, in DPPs the rules are applied according to a universal clock in a parallel manner.

A solution to the first problem can be proposed by exploiting the framework of P systems (in particular, by using DPPs), since the membrane structure is suitable to represent systems consisting of *many* regions. Details about DPPs can be found in Section 3.2, and several examples of simulated systems can be found in Section 3.3 and [161, 163, 162].

The second problem can be solved by extending the tau-leaping simulation algorithm to the modelling framework provided by DPPs. Hence, this new stochastic approach within the framework of P systems, which exploits their topological structure and other features, represents a novel tool for the modelling of multi-volume systems able to provide a *quantitative* description of the system's dynamics. Here, instead of assuming a global clock for the instantaneous and parallel application of the rules, we will select a different length for each step according to the current system state; then, during the time step, a certain number of rules will be selected and executed.

τ-DPP is a computational method which can be used to describe and perform stochastic simulations of complex biological or chemical systems. For instance, cellular pathways involving several spatial compartments (as

the extracellular ambient, the cytoplasm, the nucleus, etc.), or multicellular systems like bacterial colonies, or multi-patched ecological systems as metapopulations, are all examples of complex systems that could be investigated with τ-DPP. Since τ-DPP represents a general simulating framework for a broad range of complex systems, in the following we will use the generic terms *volume*, *object* and *rule*, to denote the compartment, or region, where the molecular species, or any other elemental "species", can be modified in some way by a biochemical or, more generally, an interspecies reaction.

The correct behaviour of the whole system is achieved by letting all volumes evolve in parallel, and by using the following strategy for the choice of time increments. At each iteration of τ-DPP, we consider the current state of each volume (determined by the current number of objects), and we calculate a time increment independently in each volume (according to the standard tau-leaping algorithm). Then, the smallest time increment is selected and used to evaluate the next-step evolution of the entire system. Since all volumes *locally* evolve according to the same time increment, τ-DPP is able to correctly work out the *global* dynamics of the system. Moreover, by adopting this procedure, the simulated evolutions of all volumes get naturally *synchronized* at the end of each iterative step. The synchronization is also necessary – and exploited together with a parallel update of all volumes – to manage the communication of objects among volumes, whenever prescribed by specific (communication) rules.

Besides the length of the local and global time increments, we need to check which kind of evolution will take place inside each volume, during the current iteration, by looking only at the volume internal state. The types of evolutionary step are those defined in the tau-leaping procedure and described in Section 1.3: a volume can evolve executing either (1) a SSA-like step, or (2) non-critical reactions only, or (3) a set of non-critical reactions plus one critical reaction.

The τ-DPP selection procedure has to consider how every volume is evolving during the actual step; then, the smallest τ generated by all volume is used to update the system.

For instance, if a volume is evolving executing only non critical reactions, but the τ generated inside it is not the smallest one of the system, then - after receiving the minimal τ from some other volume - this volume has to sample the next reaction from the set of non critical reactions.

Once the τ-DPP procedure generates a local τ, two different scenarios are possible: no volumes are evolving like SSA, or at least one volume is evolving according to SSA.

If no volumes are evolving like SSA, then the smallest τ (τ_{min}) generated inside the volumes during the current step is chosen. Then, the number of firings of the rules is sampled as the Poisson random variable $P(a_j, \tau_{min})$.

On the contrary, if there is at least one volume evolving in the SSA manner - which generates a value τ_{SSA} - then the procedure has to check

if $\tau_{min} = \tau_{SSA}$. If this is true, then τ_{min} was generated by the volume evolving according to the SSA, and the execution of the selected rule is allowed. Otherwise, if τ_{SSA} is greater than τ_{min}, then it is not possible to apply the reaction selected inside that volume, because the time required for the execution of this reaction (τ_{SSA}) is longer than the length of the current step (τ_{min}).

Formally, a τ-DPP Υ is defined as

$$\Upsilon = (V_0, \ldots, V_{n-1}, \mu, \mathcal{S}, M_0, \ldots, M_{n-1}, R_0, \ldots, R_{n-1}, C_0 \ldots C_{n-1}),$$

where:

- $V_0, \ldots, V_{n-1}$ are the volumes of the system, $n \in \mathbb{N}$, $n > 1$;
- μ is a membrane structure representing the topological arrangement of the volumes;
- $\mathcal{S} = \{X_1, \ldots, X_m\}$ is the set of objects, $m \in \mathbb{N}$, that is, the alphabet of the system;
- $M_0, \ldots, M_{n-1}$ are the multisets over $\mathcal{S}$ occurring inside the volumes $V_0, \ldots, V_{n-1}$, representing the internal state of the volumes;
- $R_0, \ldots, R_{n-1}$ are the sets of rules defined in volumes $V_0, \ldots, V_{n-1}$, respectively. A rule can be of internal or of communication type (as described below);
- $C_0, \ldots, C_{n-1}$ are the sets of stochastic constants associated to the rules defined in volumes $V_0, \ldots, V_{n-1}$.

The system Υ is defined by means of a set of n volumes organised according to the hierarchy specified by the membrane structure μ. The state of the whole system is characterised by all multisets M_i occurring inside each volume V_i ($0 \leq i \leq n-1$).

Inside the volumes, the set of rules $R_0, \ldots, R_{n-1}$ are defined along with the sets of stochastic constants $C_0, \ldots, C_{n-1}$. The stochastic constants are needed, along with a combinatorial function depending on the left-hand side of the rule (as explained in Section 1.1), to compute the probabilities of the rule applications (i.e., their propensity functions).

Each volume V_i can contain two different kinds of rules, termed *internal* and *communication* rules. An internal rule describes the modification, or evolution, of the objects inside the single volume where it is applied, while a communication rule sends the objects from the volume where it is applied to an adjacent volume (possibly modifying the form of these objects during the communication step).

More precisely, internal rules have the general form $\alpha_1 X_1 + \alpha_2 X_2 + \cdots + \alpha_m X_m \rightarrow \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_m X_m$, where $X_1, \ldots, X_m \in \mathcal{S}$ are distinct

object types and $\alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_m \in \mathbb{N}$. For instance, $X_1, \ldots, X_m$ can correspond to molecular species, and, in this case, $\alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_m$ represent stoichiometric coefficients. The objects appearing in the left-hand side of the rule are called *reagents*, while the objects on the right-hand side are called *products*. Note that, usually, we will consider the case where (at most) three objects appear in the reagents group. The rational behind this is that we require biochemical reactions to be (at most) of the third-order, since the simultaneous collision and chemical interaction of more than three molecules at a time, has a probability to occur close to zero in real biochemical systems. Moreover, the interaction among more than three molecules can be modelled by using a set of successive reactions with lower order.

When dealing with communication rules inside a volume, besides defining the sets of reagents and products, it is necessary to specify the target volume where the products of this rule will be sent[1]. Formally, a communication rule has the form[2] $\alpha_1 S_1 + \alpha_2 S_2 + \cdots + \alpha_m S_m \rightarrow (\beta_1 S_1 + \beta_2 S_2 + \cdots + \beta_m S_m, tar)$, where $S_1, \ldots, S_m \in \mathcal{S}$ are distinct object types, $\alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_m \in \mathbb{N}$, and tar can be equal to:

- *out*: this means that the products of the rule are "sent outside" the source volume (that is, the region where the rule is applied), to the adjacent outer volume;
- in_{label}: this means that the products of the rule are "sent inside" the volume with the label specified by the target. These rules are only allowed if the target volume is placed inside the source membrane, and the two volumes are adjacent (that is, there exists no other volume placed between the source and the target volume).
- *in*: this means that the products of the rule are nondeterministically sent to any of the volumes placed inside the source membrane. This kind of rule can be used instead of a set of rules with specific targets in_{label} (one rule for each inner volume).

Communication rules are considered special rules for what concerns the time increment (τ) selection procedure, applied in the first stage of the τ-DPP algorithm. For internal rules, indeed, this procedure is exactly the same as the tau-leaping algorithm [26], where the length of the step is computed by bounding the variation of the molecular quantities. Namely, in order to correctly evaluate the simulation time increment and to describe the behaviour of the system with a good approximation, we need to choose the "largest" value of τ that also satisfies the leap condition [26].

[1] This definition can be easily extended in order to assign a different target volume to each object appearing in the set of products.

[2] The condition that at most three objects appear as reagents is usually required also for communication rules.

On the contrary, the variation due to communication rules implies a change in the quantities of objects inside *two* different volumes: the reagents inside the source volume, and the products sent to the destination volume. To correctly estimate the value of τ when dealing with communication rules, instead of limiting the variation of both reagents and products (as it is done for internal rules), we only consider the variation of the reagents inside the source volume (that is, we only look at the left-hand side of the communication rule). Indeed, the value of τ is independent from any objects that has to be communicated, since these products will be received in the target volume only at the end of the iteration step (see Section 4.2). For this reason, for the τ selection procedure, the right-hand side of a communication rule is neither considered in the source, nor in the target volume. Obviously, the communicated objects will contribute to the update of the system state, which takes place at the end of the iteration step, and will be therefore considered to determine the state of the target volume for the next iteration step.

4.2 τ-DPP algorithm

The τ-DPP algorithm proceeds by executing iterative steps to simulate the evolution of the entire system. Each step consists of several substeps that are executed *independently* and *in parallel* for each single volume V_i, $0 \leq i \leq n-1$, of the system. In the following description, the algorithm execution naturally proceeds according to the order of instructions, when not otherwise specified by means of "go to" commands.

1. Initialization: load the description of each volume V_i, given by the multiset M_i, the set of internal and communication rules and their respective stochastic constants, the set of targets for each communication rule;

2. Compute the propensity function a_j of each rule R_j, $0 \leq j \leq m-1$, and evaluate the sum of all the propensity functions in V_i, $a_0 = \sum_{j=0}^{m-1} a_j$. If $a_0 = 0$, then go to *Step 3*, otherwise go to *Step 5*;

3. Set τ_i, the length of the step increment in volume V_i, to ∞;

4. Wait for the communication of the smallest time increment $\tau_{min} = \min\{\tau_0, \ldots, \tau_{n-1}\}$ among those generated independently inside all volumes $V_0, \ldots, V_{n-1}$, during the current iteration, then go to *Step 13*;

5. Generate the step size τ_i according to the internal state, and select the way to proceed in the current iteration (i.e. SSA-like evolution, or tau-leaping evolution with non-critical reactions only, or tau-leaping evolution with non-critical reactions and one critical reaction), using the selection procedure defined in [26];

6. Wait for the communication of the smallest time increment $\tau_{min} = \min\{\tau_0, \ldots, \tau_{n-1}\}$ among those generated independently inside all volumes, during the current iteration. Then:

   - if the evolution is SSA-like and the value $\tau_i = \tau_{SSA}$ generated inside the volume is greater than τ_{min}, then go to *Step 7*,
   - if the evolution is SSA-like and the value $\tau_i = \tau_{SSA}$ is equal to τ_{min}, then go to *Step 10*,
   - if the evolution is tau-leaping with non-critical reactions plus one critical reaction, and the value $\tau_i = \tau_{nc1c}$ is equal to τ_{min}, then go to *Step 11*,
   - if the evolution is tau-leaping with non-critical reactions plus one critical reaction, and the value $\tau_i = \tau_{nc1c}$ is greater than τ_{min}, then go to *Step 12*,
   - if the evolution is tau-leaping with non-critical reactions only ($\tau_i = \tau_{nc}$), then go to *Step 12*;

7. Compute $\tau_{SSA} = \tau_{SSA} - \tau_{min}$;

8. Wait for possible communication of objects from other volumes, by means of communication rules. If some object is received, then go back to *Step 2*, otherwise go to *Step 9*;

9. Set $\tau_i = \tau_{SSA}$ for the next iteration, then go back to *Step 6*;

10. Using the SSA strategy [74], extract the rule that will be applied in the current iteration, then go to *Step 13*;

11. Extract the critical rule that will be applied in the current iteration;

12. Extract the set of non-critical rules that will be applied in the current iteration;

13. Check if the execution of the selected rules leads to an unfeasible state, namely, there are negative amounts of molecules inside the volume. If this is the case, then reduce τ_{min} by half and go to *step 6*. Otherwise go to *step 14*;

14. Update the internal state by applying the extracted rules (both internal and communication) to modify the current number of objects, and then check for objects (possibly) received from the other volumes.

15. If the termination criterion is satisfied, then end the simulation. Otherwise go back to *Step 2*.

The initialization phase of the algorithm is executed in parallel inside every single volume of the system, where distinct rules and amounts of objects may appear. The computation of the propensity functions a_j of the rules R_j is performed exploiting the expression presented in [74]. In step 2, we check whether the sum a_0 of all the propensity functions of the volume is equal to zero. If so, then no rule can be executed inside this volume; in this case, we set $\tau_i = \infty$ and let the volume wait for the current τ_{min} value, chosen among the time increments computed inside the other volumes. This step is necessary for the synchronization between this volume and the other ones where, possibly, some rules will be applied. So doing, during the update at the final step of the algorithm, it will also be possible to check whether the volume is the target of some communication rules applied in other volumes (that is, whether it will receive new objects), and hence properly update its internal state.

On the other hand, if $a_0 > 0$, the value of τ_i is computed inside the volume considering only its internal state (this is done exploiting the first part of the tau-leaping procedure presented in [26]). The τ_i computation also implies the selection of the kind of evolution for the current iteration inside each volume V_i, independently from the kind of evolution selected in the other volumes.

Once every volume of the system has computed its τ_i value, the smallest one is selected and used to generate the evolution of the *whole* system (step 6 of the algorithm) during the current iteration. This means that each volume will not evolve according to the internally computed τ_i, but according to the common value τ_{min}. The rational behind this is that, we let all volumes proceed along a common time line, therefore avoiding paradoxical situations where one volume will execute rules that take place in the future or in the past time of another volume.

When every volume has received the common value τ_{min}, according to the evolution strategy selected at step 5 of the algorithm, it extracts the rules that will be applied in the final stage of the iteration. If the evolution of the volume is governed by the SSA strategy, we have to check if τ_{min} is equal to τ_i (here called τ_{SSA}). If so, it means that the chosen τ_{min} value was actually generated inside this volume, and that the current iteration is "long enough" to allow the application of *one* rule inside this volume. On the other hand, if $\tau_{min} < \tau_{SSA}$, then it is not possible to apply any rule inside this volume. For this reason we only have to update the value of τ_i (step 7 of the algorithm).

Afterwards, the volume verifies for possible incoming objects: if something is received from other volumes, then the internal state of the volume is changed, and in the next iteration the volume will compute new values of the propensity functions and a new τ_i value. On the contrary, if the volume does not receive anything from its adjacent volumes, then its internal state remains unchanged. For this reason, the τ_i value used in the next iteration

will be the current τ_{SSA} value (updated during the step 7 of the algorithm). Notice that the value $\tau_i = \tau_{SSA} - \tau_{min}$ is used again in the next iteration because, in the case τ_i is the smallest time increment of the whole system, then the volume will be able to complete the execution of one rule.

The situation is easier for the other kinds of evolution. If the volume is executing a tau-leaping step with the application of a set of non-critical reactions and one critical reaction, we have to check if $\tau_i = \tau_{min}$. If this is true, besides the execution of the non-critical reactions extracted from the poissonian distributions (we refer to [26] for details), it is possible to execute one critical reaction. Indeed, the value τ_i computed inside the volume corresponds to the time needed to execute such critical reaction (this is the reason why its execution is allowed). On the contrary, if $\tau_i > \tau_{min}$, the volume will execute non-critical reactions only. Finally, if the volume is evolving according to the tau-leaping procedure with the execution of non-critical reactions only, we use τ_{min} for the sampling of the number of rules that will be executed (according to the strategy described in [26]).

During step 13, the algorithm checks if the execution of the selected reactions (both internal and communication) leads to an unfeasible system state, where negative amounts of objects appear. In this case, the strategy to avoid this problem consists in reducing the value of τ_{min} by half and go back to step 6 of the algorithm. So doing, a new set of reactions (possibly smaller than the previous one) will be selected. The condition about negative population is checked until an appropriate set of reactions is selected, halving each time the length of the time step.

Finally, the state of the system is updated. Every volume executes the selected reactions, both internal and communication, hence the number of objects is properly updated also by means of the objects (possibly) received from the other volumes. Note that the internal states of all volumes are updated in parallel and, thanks to the choice of the common time increment τ_{min}, also in a synchronized fashion.

The algorithm takes a computational time equal to that of the original tau-leaping procedure for the simulation of a single volume, since the τ calculation strategy is the same. Hence, the time required to execute each iteration is proportional to $2M$ (where M is the number of reactions of the system). For a generic system composed by N volumes, the computational time becomes $2MN$, because it increases with the number of volumes itself. The complexity of τ-DPP is different from other algorithms, like the NSM, where a single volume evolves during each iteration. Indeed, in τ-DPP the internal state of each volume has to be updated and, in order to compute the length of the time step, the contribution of the entire system is required.

On the other hand, the τ-DPP strategy can be easily coded for parallel architectures, where a one-to-one correspondence between volumes and processes or threads, can be obtained. In this case, the computational time of the algorithm scales with the number of "parallel processes" executed, with

a little overhead due to the communication among volumes.

4.3 A test case for τ-DPP

In this section we present a simple chemical system used to test the accuracy and the correctness of the τ selection procedure and of the communication and synchronisation between volumes of τ-DPP. This test case consists of two consecutive reactions, where the product of the first one is the reactant of the second one:

$$r_1 : A \xrightarrow{0.1} B$$
$$r_2 : B \xrightarrow{0.025} C$$

The consecutive reactions systems has been implemented by using two different configurations: one consisting in a single volume where both reactions appear, which has been simulated by means of tau-leaping algorithm. The other configuration is composed by two volumes V_1 and V_2, each volume containing a single reaction (r_1 in V_1, r_2 in V_2), and it has been simulated by using the τ-DPP algorithm. In this last configuration, the reactions have been adapted to the new structure, hence they are both defined as communication reactions. In particular, reaction r_1 has now the form $A \rightarrow B, in_2$ and r_2 is $B \rightarrow C, in_1$; moreover, in both configurations of the consecutive reaction system we use the same set of stochastic constants.

Figure 4.1 shows the two dynamics obtained using the tau-leaping and the τ-DPP algorithms. The comparable results prove the correctness and reliability of τ-DPP.

4.4 τ-DPP with size associated to volumes and objects

The current variants of membrane systems used for the modelling of biochemical systems provide a description where volumes can contain up to an infinite number of molecules, since the sizes of the structure components and of the objects contained within the volumes are not considered.

Moreover, communication channels are limited to adjacent volumes, that is, it is possible to define rules that send objects between adjacent volumes. In particular, in P systems with a tree–like membrane structure, the communication is usually permitted only from a volume to another one immediately inside or outside the first one and vice–versa. On the other hand, also in the case of tissue P systems [121] (or tP systems), the communication of objects is allowed only between adjacent membranes, and it is achieved by using the "synapses" defined between nodes (recently, different variants on P systems

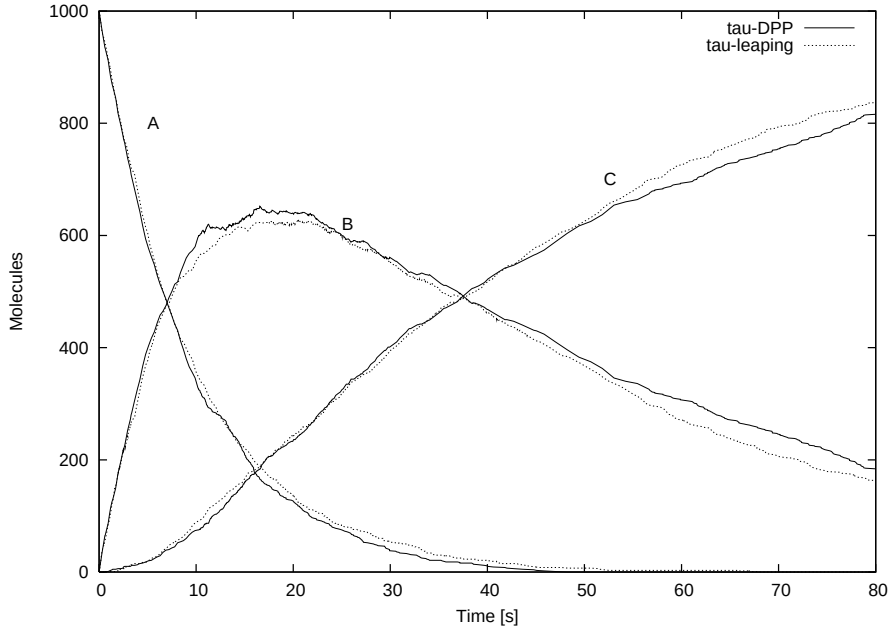


Figure 4.1: Comparison between the dynamics of the consecutive reaction system obtained by means of tau-leaping (dashed lines) and τ-DPP (solid lines).

have been introduced to better model economic or socio-economic processes, databases, operating systems, etc., in which different communication strategies are used, like *Hyperdag P systems* [148], or variants like that presented in [155]).

For what concerns the membrane structure, either variant of P systems use only a tree–like or a tissue–like structure, while hybrid structures are not considered. For instance, the description of tissues where nodes can have a complex internal structure, or tree–like systems where membranes can enclose a tissue, have never been defined.

We briefly report here the basic notions of *tP systems* [121]. tP systems have been defined to describe a tissue–like architecture, where cells are placed in the nodes of a (directed) graph, and objects are communicated along the edges of the graph. These communication channels are called synapses. Moreover, the communication of objects can be achieved both in a replicative and non–replicative manner, that is, objects can be sent to all adjacent cells or to one adjacent cell only, respectively.

In general, the structure of a tP system is composed by elementary membranes, namely, each node of the system is represented by a membrane that does not contain other membranes. Furthermore, the communication of objects is allowed, as in standard P systems, only to/from adjacent membranes. We refer the interested reader to [65, 157, 4, 116, 120] for further

information and examples about tP systems.

Hereafter, we present a different variant of τ-DPP, called Sτ-DPP [29], where we associate "measures" which represent the size of volumes and objects, we use tP systems [121] to describe the topological organisation of the membranes and to denote the possible communication channels of the system, and we exploit the simulation strategy described in Section 4.1 to describe the behaviour of the system. In the following, we will refer without distinction to membranes or volumes.

To be more precise, in Sτ-DPP we associate a measure to membranes and objects, representing the "size" of the volumes where the computation occurs and the volume of each object, respectively. Both the sizes of membranes and objects are useful to describe any real system where it is important to avoid the infinite accumulation of objects inside a compartment, which is very important in chemical system and cannot be achieved by simply bounding the "capacity" of the membranes or by limiting the maximum number of each kind object allowed inside a particular volume, since each object can have a different size. Moreover, in the framework of Sτ-DPP, the system's structure is independent from the communication channels between membranes, hence, two different graphs are used in the description: the first one denotes the membrane structure, while the second graph specifies the connections between membranes which allow the communication of objects. Furthermore, the membrane structure can be hybrid (since it combines tree-like P systems with tissue P systems), and the communication can be performed between non adjacent membranes, to denote privileged pathways between membranes. This characteristic of Sτ-DPP takes inspiration from a specific component of living cells, called *microtubule* [167], with the aim of reproducing its role as intracellular "highway" for the transport of other cellular components, such as vesicles and proteins.

Sτ-DPP

In this section we will give a detailed description of Sτ-DPP, providing a basic definition and the algorithm used to simulate a system's dynamics. As already said, Sτ-DPP is obtained combining the structure definition of tP systems with the simulation strategy used in τ-DPP. Here, nodes are arranged in a tissue–like fashion, but each of them can have a complex internal hierarchy, organised in a tree–like structure. Moreover, in this new variant we consider sizes for both membranes and objects, and the rules defined inside each membrane will be enabled only in the case there is sufficient free space in the membrane where the rule is applied, for instance, to "create" new objects or to receive objects from other volumes. The size considered here can be used in the modelling and simulation of biochemical systems where diffusive processes play an important role, and it is necessary to avoid the unlimited accumulation of objects in a region of finite size.

In order to correctly describe the hierarchy of complex nodes of the system we first need a graph representing the topology of the membranes. In particular, this graph can have undirected edges to indicate that two membranes are placed at the same hierarchical level (as in the standard definition of tP systems [121]). On the other hand, directed edges of the graph are used to denote that the "source" membrane contains the "target" membrane.

A second directed graph is needed to represent the communication channels among membranes. Clearly, the arrows of the edges indicate the direction of the (permitted) flow of objects between different compartments. Note that, this communication graph can contain edges that are not indicated in the graph which describes the membrane structure. The meaning of these particular edges is to represent communication channels that connect non adjacent membranes. Using these arcs, it is then possible to create privileged pathways of communication between membranes.

The features of Sτ-DPP can be exploited to represent (among other real life systems) reaction–diffusion systems [47], namely, mathematical models which capture the dynamics of a set of substances involved in a number of chemical reactions, considering both the temporal and spatial dimension. In this case, the membrane structure can be used to represent a reaction volume as the composition of a number of finite size subvolumes, and the communication graph will describe the diffusion directions through the system.

Definition

A Sτ-DPP with sizes associated with membranes and objects is defined as

$$\Upsilon = (\mathcal{V}, \mathcal{T}_G, \mathcal{C}_G, \mathcal{S}, \mathcal{M}, \mathcal{R}, \mathcal{C}, \mathcal{D}_X, \mathcal{D}_V),$$

where:

- $\mathcal{V} = \{V_0, \ldots, V_{n-1}\}$ is the set of the volumes V_i of the system;
- $\mathcal{T}_G = (\mathcal{V}, A_\mathcal{T})$ is a graph representing the topological arrangement of the volumes in $\mathcal{V}$ and $A_\mathcal{T} = A_\mathcal{T}^u \cup A_\mathcal{T}^d$ is the set of the arcs (V_l, V_k) which describes the arrangement of volumes. In particular, $A_\mathcal{T}^u$ is the subset of undirected arcs representing the connections between membranes placed at the same level in the membrane structure. On the contrary, $A_\mathcal{T}^d$ is the set of directed arcs (V_l, V_k) (where $V_k \in V_l$) which specifies the inclusion relations between volumes. We also define the set of the volumes enclosed in V_i as $a_\mathcal{T}(V_i) = \{V_l \mid V_l \in \mathcal{V},\ (V_i, V_l) \in A_\mathcal{T}^d\}$;
- $\mathcal{C}_G = (\mathcal{V}, A_\mathcal{C})$ is a directed graph representing the connections (communication channels) among the volumes in $\mathcal{V}$. $A_\mathcal{C}$ is the set of directed

arcs (V_l, V_k) (from V_l to V_k) which denote the presence of communication channels between volumes and specify in which directions the flow of objects is allowed;

- $\mathcal{S} = \{X_1, \ldots, X_M\}$ is the set of molecular species, that is, the alphabet of the system;
- $\mathcal{M} = \{M_0, \ldots, M_{n-1}\}$, is the set of the multisets occurring inside the membranes $V_0, \ldots, V_{n-1}$, representing the internal state of the volumes. Each multiset M_i $(0 \leq i \leq n-1)$ is defined over $\mathcal{S}$;
- $\mathcal{R} = \{R_0, \ldots, R_{n-1}\}$ is the set of the sets of rules defined in volumes $V_0, \ldots, V_{n-1}$, respectively. A rule can be of internal or of communication type (as described below);
- $\mathcal{C} = \{C_0, \ldots, C_{n-1}\}$ is the set of the sets of stochastic constants associated to the rules defined in volumes $V_0, \ldots, V_{n-1}$;
- $\mathcal{D}_X = \{D_{X_1}, \ldots, D_{X_M}\}$, with $D_{X_j} \in \mathbb{R}^+$, is the set of the sizes of the molecular species $X_1, \ldots, X_M$, respectively;
- $\mathcal{D}_V = \{D_{V_0}, \ldots, D_{V_{n-1}}\}$, with $D_{V_i} \in \mathbb{R}^+$, is the set of the sizes of the volume $V_0, \ldots, V_N$, respectively.

The multiset M_i, describing the state of volume V_i $(i = 0, \ldots, n-1)$, is defined as $M_i = (m_0, \ldots, m_M)$ where m_j denotes the number of molecules of the species X_j occurring inside V_i $(j = 1, \ldots, M)$.

Given the internal state M_i of a membrane V_i together with the species sizes in $\mathcal{D}_X$, it is possible to define the *occupied volume* in V_i as:

$$O(V_i) = \sum_{j=1}^{M} (m_j \cdot D_{X_j}) + \sum_{V_l \in a_{\mathcal{T}}(V_i)} D_{V_l}, \tag{4.1}$$

$O(V_i)$ denotes the volume occupied by the objects currently present inside volume V_i.

Hence, it is possible to define the value of the *free space* in V_i as:

$$F(V_i) = D_{V_i} - O(V_i) \tag{4.2}$$

Note that, at each rule execution (internal or communication), the free space value has to be updated. The update operation adds to the free space value the "volume" of the objects consumed or sent by the rule and subtracts the "volume" of the objects produced by the rule or received from other membranes. Internal rules have the general form $\alpha_1 X_1 + \alpha_2 X_2 + \cdots + \alpha_M X_M \rightarrow \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_M X_M$, after the execution of this kind of rules, the free space in V_i is updated as $F(V_i) = F(V_i) - \sum_{j=1}^{M} (\beta_j - \alpha_j) \cdot D_{X_j}$. On the contrary, a communication rule, have the generic form

$\alpha_1 X_1 + \alpha_2 X_2 + \cdots + \alpha_M X_M \rightarrow (\beta_{1,1} X_1 + \cdots + \beta_{M,1} X_M, tgt_1) + (\beta_{1,2} X_1 + \cdots + \beta_{M,2} X_M, tgt_2) + \cdots + (\beta_{1,N} X_1 + \cdots + \beta_{M,N} X_M, tgt_N)$. In this case we need to update the free space of V_i (i.e., the membrane where the reaction is applied) as $F(V_i) = F(V_i) + \sum_{j=1}^{M} \alpha_j \cdot D_{X_j}$ and the free space of each target volume V_{tgt_k} indicated by the rule: $F(V_{tgt_k}) = F(V_{tgt_k}) - \sum_{j=1}^{M} \beta_{j,k} D_{X_j} \geq 0$.

In order to obtain a correct description of the system's dynamics, we need to check if a rule r_μ (internal or communication) is applicable. Therefore, we need to compute the effect of a rule on the free space of the volume affected by the rule. It is clear that a rule can be executed only if the free space of the volume, after the rule application, is greater or equal to zero. The rule applicability is computed differently for internal and communication rules. Given an internal rule occurring inside volume V_i, we need to check if, after the rule execution, $F(V_i) \geq 0$. For what concerns a communication rule r_μ, we need to check all the free space of the target volumes indicated by the rule: $\forall\ tgt_l$ of r_μ, $F(V_{tgt_l}) \geq 0$, where the values β_j are the stoichiometric coefficients of the molecular species associated with V_{tgt_l}.

Note that, using a strategy based on the tau-leaping algorithm to describe the behaviour of the system, at each iteration step, a certain number of rules is applied in parallel. Hence, the applicability of the entire set of selected rules need to be verified. This operation is realised by computing the free values of each volume V_i considering the contribution of all the selected rules; if the values of $F(V_i)$ is greater or equal to zero (for each volume), then the execution is allowed; otherwise a new set of rules need to be selected.

The algorithm

The Sτ-DPP algorithm is based on the original τ-DPP procedure presented in Section 4.2. The differences between the two algorithms regard the specific operations defined in order to handle the size of membrane and objects, to compute the free space of each membrane and to check the applicability of the selected rules.

In particular, at the beginning of the computation the Sτ-DPP algorithms needs to initialise the volume by loading all the information related to the size of membranes and objects in order to compute the initial value of the free space. The other step of the iterative cycle are the same as in τ-DPP, since the "sizes" are not involved in the computation of the propensity functions and in the rules selection.

On the other hand, in the final step of Sτ-DPP, we need to verify if the execution of the selected set of rules leads to an unfeasible state of the system, with negative values of the free spaces of the membranes. This phase of the algorithm is realised by computing, inside each volume, the new value of the free space considering the contribution of the selected rules, both internal and communication. If there is at least one volume whose free space value is negative, then the execution of the rules is not allowed. In

this case, the value of the length of the step τ is reduced by half, and a new (possibly smaller) set of rules is selected. This operations are repeated until the condition on the free space is satisfied (i.e., no negative free space values are present). Note that this strategy is the same used when negative amounts of objects appear in the system.

Finally, the system is updated according to the selected rules by modifying the objects amounts and the values of the free space of each volume.

The other features introduced in Sτ-DPP regarding the possibility to send objects to non-adjacent membranes and the definition of hybrid membrane structure do not affect the simulation procedure, since they are implicitly considered in the algorithm.

4.5 A test case for Sτ-DPP

In this section we present a system with preferential communication pathways. In particular, we define a model in which it is present a so called *microtubule* [167]. The microtubule is a sort of intracellular "highway" for the transport of other cellular components, such as vesicles and proteins. In order to define a preferential pathway, we exploit the communication between non adjacent membranes to move objects in particular directions. We consider the membrane system Υ represented in Figure 4.2, where:

- $\mathcal{V} = \{V_0, \ldots, V_7\}$;
- $\mathcal{T}_G = (\mathcal{V}, A_\mathcal{T})$, where $A^u_\mathcal{T} = \{(V_2, V_3), (V_4, V_5), (V_6, V_7)\}$, and $A^d_\mathcal{T} = \{(V_0, V_1), (V_1, V_2), (V_1, V_3), (V_3, V_4), (V_3, V_5), (V_5, V_6), (V_5, V_7)\}$;
- $\mathcal{C}_G = (\mathcal{V}, A_\mathcal{C})$, $A_\mathcal{C} = \{(V_0, V_1),\ (V_1, V_0),\ (V_1, V_2),\ (V_1, V_3),\ (V_3, V_1),\ (V_3, V_2),\ (V_3, V_4),\ (V_3, V_5),\ (V_5, V_3),\ (V_5, V_4),\ (V_5, V_6),\ (V_5, V_7),\ (V_7, V_5),\ (V_6, V_7)\}$;
- $\mathcal{S} = \{X_1, X_2\}$;
- $\mathcal{M} = \{M_0, \ldots, M_7\}$, $M_0 = \{X_1^{10^5}, X_2^{10^5}\}$, $M_3 = M_4 = \ldots = M_7 = \emptyset$;
- $\mathcal{R} = \{R_0, \ldots, R_7\}$, $R_0 = \{r_{0,0}, r_{0,1}\}$, $R_1 = \{r_{1,0}, \ldots, r_{1,4}\}$, $R_2 = \{r_{2,0}\}$, $R_3 = \{r_{3,0}, \ldots, r_{3,2}, \ldots, r_{3,4}\}$, $R_4 = \{r_{4,0}\}$, $R_5 = \{r_{5,0}, \ldots, r_{5,4}\}$, $R_6 = \{r_{6,0}\}$, $R_7 = \{r_{7,0}, \ldots, r_{7,4}\}$;
- $\mathcal{C} = \{C_0, \ldots, C_7\}$, where the value of all the stochastic constants is set to 1;
- $\mathcal{D}_X = \{1, 1\}$;
- $\mathcal{D}_V = \{10^6, 11 \cdot 10^4, 10^4, 10^6, 8 \cdot 10^4, 5 \cdot 10^4, 10^4, 2 \cdot 10^4\}$.

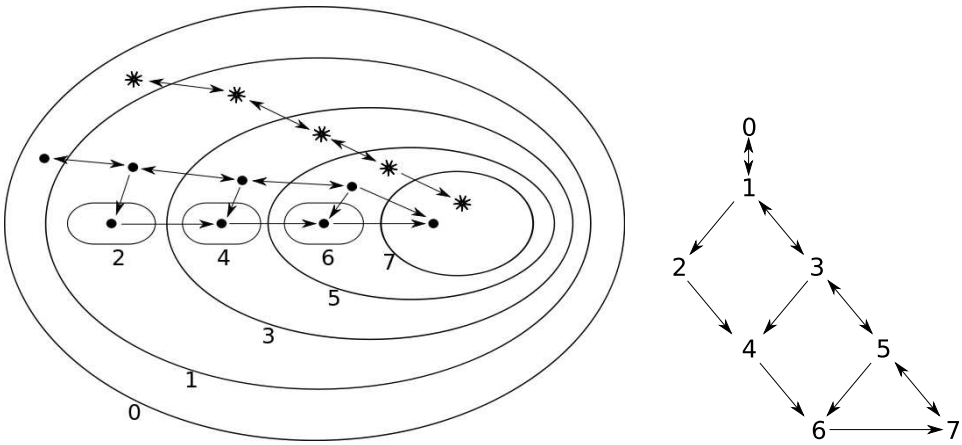


Figure 4.2: The membrane system Υ with communication channels between non adjacent membranes, in the graphical representation (left side), the molecular species X_1 is indicated as a dot, while X_2 is denoted as a star, the arrows indicate the allowed flow of information of the two molecular species. On the right side, the communication graph $\mathcal{C}_G$, is reported.

Table 4.1: Rules of the membrane system Υ. The stochastic constants associated to the rules are all set to 1.

Reaction		**Reaction**	
$r_{0,0}:$	$X_1 \to (X_1, 1)$	$r_{3,2}:$	$X_1 \to (X_1, 4)$
$r_{0,1}:$	$X_2 \to (X_2, 1)$	$r_{3,3}:$	$X_1 \to (X_1, 5)$
$r_{1,0}:$	$X_1 \to (X_1, 0)$	$r_{3,4}:$	$X_2 \to (X_2, 5)$
$r_{1,1}:$	$X_2 \to (X_2, 0)$	$r_{4,0}:$	$X_1 \to (X_1, 6)$
$r_{1,2}:$	$X_1 \to (X_1, 2)$	$r_{5,1}:$	$X_1 \to (X_1, 3)$
$r_{1,3}:$	$X_1 \to (X_1, 3)$	$r_{5,2}:$	$X_2 \to (X_2, 3)$
$r_{1,4}:$	$X_2 \to (X_2, 3)$	$r_{5,3}:$	$X_1 \to (X_1, 6)$
$r_{2,0}:$	$X_1 \to (X_1, 4)$	$r_{5,4}:$	$X_1 \to (X_1, 7)$
$r_{3,0}:$	$X_1 \to (X_1, 1)$	$r_{5_5}:$	$X_2 \to (X_2, 7)$
$r_{3,1}:$	$X_2 \to (X_2, 1)$	$r_{6,0}:$	$X_1 \to (X_1, 7)$

Υ represents a simplified version of a "cellular" system which describes the "movement" of molecules X_1 and X_2 from the "extracellular space" (membrane V_0) to the "nucleus" (membrane V_7), passing through nested regions of the "cytoplasm", (membranes V_1, V_3, V_5), or through a "microtubule" (membranes V_2, V_4, V_6). The rules listed in Table 4.1 describe the diffusive processes (through the membranes of the system) related to X_1 and X_2. Note that, only X_1 molecules can "enter" into the microtubule regions, and then, they can move only towards membrane V_7 (the nucleus). On the contrary, in the other regions, that is, outside the microtubule, the diffusion is enabled in every direction for both molecules X_1 and X_2.

Hereafter we report the results obtained by simulating the membrane system Υ. In the first set of simulations we compared the behaviour of the system as reported in Figure 4.2 with the behaviour obtained with a different configuration (Figure 4.3), namely, we "sealed" the microtubule in such a way that the molecules X_1 can enter into it only from membrane V_1 (into membrane V_2).

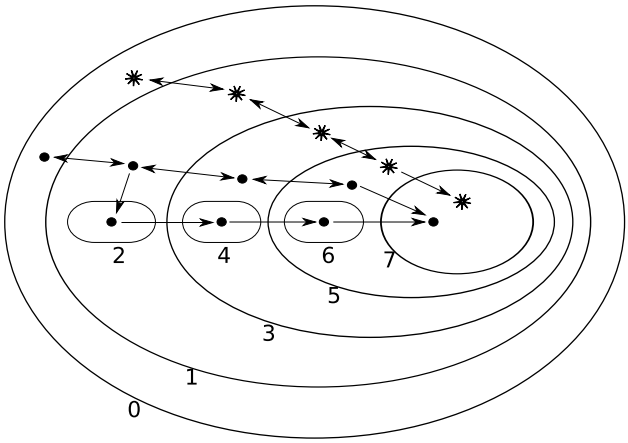


Figure 4.3: The membrane system Υ with "sealed" microtubule. This configuration allows the movement of molecules X_1 into the microtubule only from membrane V_1.

In Figure 4.4, the dynamics of membrane V_0 and V_7 is shown. As expected, the diffusion of molecules X_2 (black line) is slower than that of molecules X_1 in both configurations: with the microtubule accessible from any membrane (blue line) or with the "sealed" microtubule (red line).

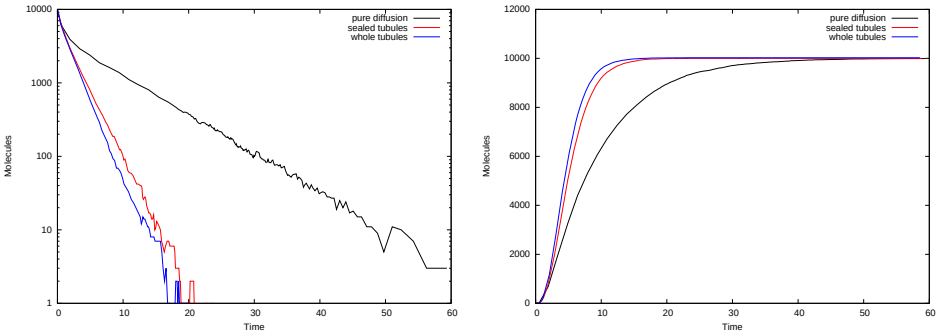


Figure 4.4: Dynamics of membrane V_0 (left) and V_7 (right). In both graphs, the dynamics of X_2 (black line) and X_1 with the completely open microtubule (blue line) and the sealed microtubule (red line), is reported.

Figure 4.5 shows the dynamics of membranes V_1, V_2, V_3 and V_6 of the system Υ. Inside membrane V_1 and V_3 (top left and right of Figure 4.5)

there is a greater amount of X_2 with respect to molecules X_1, since the latter can diffuse towards membrane V_7 passing through the microtubule. On the other hand, inside the membranes which compose the microtubule, the dynamics obtained with the two different configurations are comparable for what concerns membrane V_2; in the other membranes, and in the configuration with sealed microtubule, the quantity of X_1 is lower, since this kind of molecules cannot enter into the microtubule from any membrane of the cytoplasm and they therefore move through membranes V_3 and V_5, resulting in a slower diffusion towards membrane V_7.

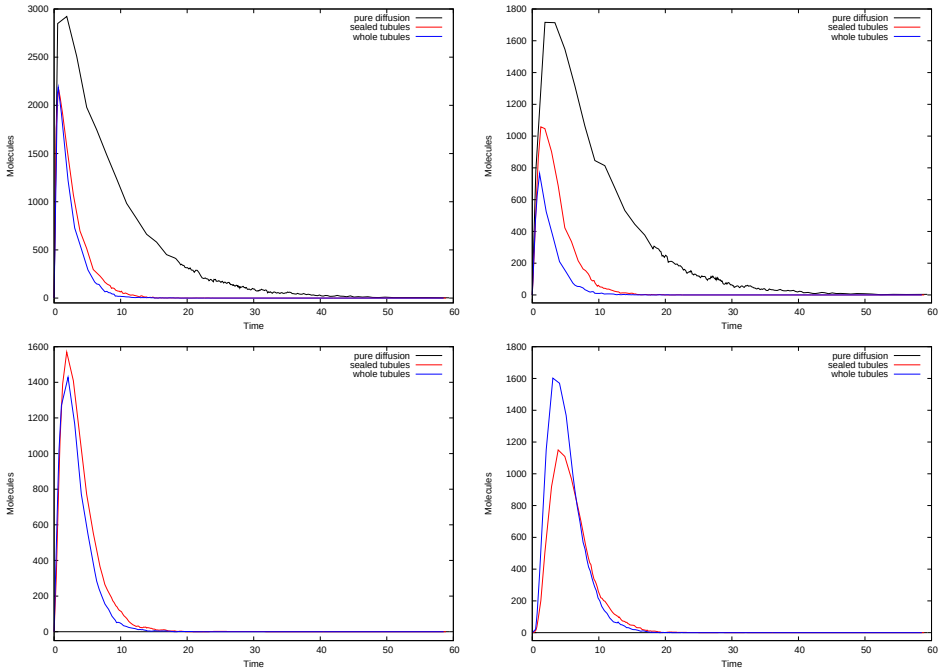


Figure 4.5: Dynamics of membrane V_1 and V_3 (top left and right) and membrane V_2 and V_6 (bottom left and right). In the graphs, the dynamics of X_2 (black line) and X_1 with the completely open microtubule (blue line) and the sealed microtubule (red line), is reported. Clearly, the quantity of X_2 inside membranes V_2 and V_6 is always zero because this molecular species cannot enter into the microtubule.

In the second set of simulations, we considered a configuration where the size of one membrane of the microtubule (namely, membrane V_4) is much more smaller than the size used in the initial configuration. In particular the size of membrane V_4 has been modified from $8 \cdot 10^4$ to 100.

The results of the simulations show that the diffusion towards the nucleus V_7 of the cell is slower in the configuration with the bottleneck represented by the reduced size of membrane V_4 (Figure 4.6).

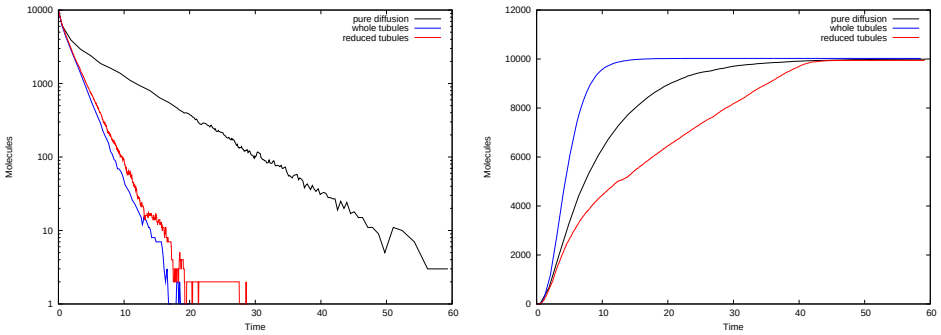


Figure 4.6: Dynamics of membrane V_0 (left) and V_7 (right). In both graphs, the dynamics of X_2 (black line) and X_1 with the completely open microtubule (blue line) and the "reduced" microtubule (red line), is reported.

Figure 4.7 shows the dynamics of the membranes representing the microtubule: V_2 (top), V_4 (bottom left) and V_6 (bottom right). As expected, the reduced size of membrane V_4 leads to an accumulation of X_1 inside membrane V_2 since the molecules enter into this membrane but they cannot move because membrane V_4 is full. As a result, also the quantity of molecules X_1 inside membrane V_6 is low, because of the slower communication of molecules from membrane V_4.

4.6 Implementation of the algorithms

In the implementation of this kind of stochastic algorithms, the use of suitable data structures to encode the information of the system to simulate is very important because it directly affects the computation time required for the simulations.

The set of chemical reactions can be represented by means of matrices, one for the left-hand side and one for the right-hand side of the reactions (which encode the stoichiometric matrix of the system). Two different matrices are required because only the left-hand side of each reaction is used to compute the corresponding propensity function, while the right-hand side is considered during the update operations.

In general, biochemical systems are composed of unimolecular and bimolecular reactions over a large set of molecular species, hence, the resulting matrices are sparse. If these data structure are codified by means of multi–dimensional arrays, then the reading operations might be very time consuming (proportional to MN, with M as number of reactions and N number of chemical species). Therefore, the set of reactions has to be codified through *linked lists*, resulting in faster reading operations (proportional to M). The other information of the system, such as the set of molecular species, the set of stochastic constants, the targets of the rules and the size

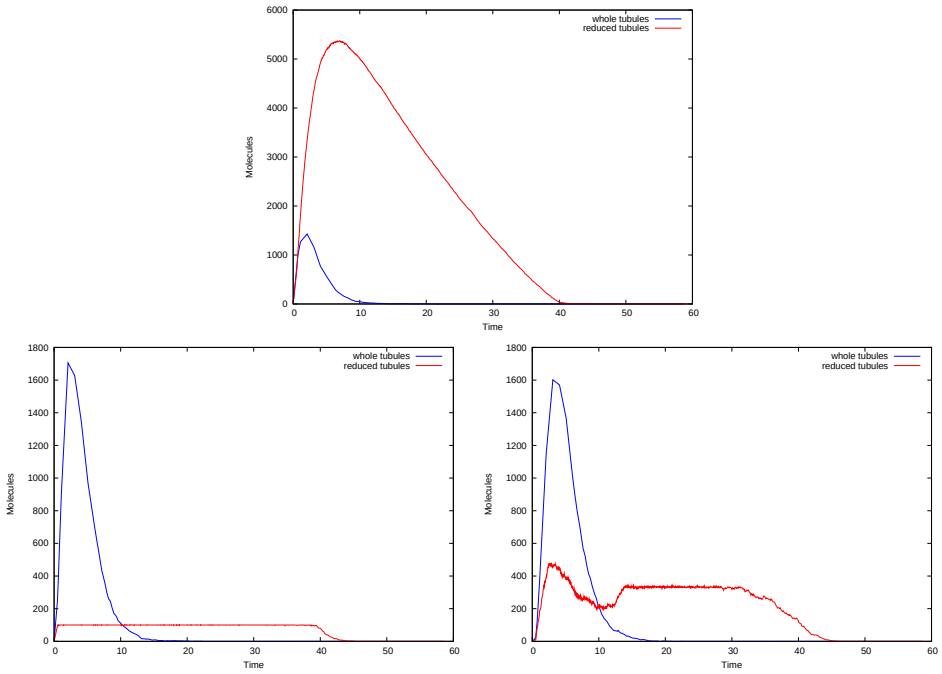


Figure 4.7: Dynamics of membrane V_2 (top) and membrane V_4 and V_6 (bottom left and right). In the graphs, the dynamics of X_1 with the completely open microtubule (blue line) and the "reduced" microtubule (red line), is reported.

of volumes and objects (in the case of Sτ-DPP), can be encoded as vectors, since all the reading operations related to these data are usually codified by means of iterative cycles which sequentially read all the values.

Besides the definition of the data structure needed to handle the information about the system to simulate, another important issue regards the programming language which can be used to implement the algorithms. For what concerns τ-DPP and Sτ-DPP, the most important feature required is the efficiency; among the well-known and most used programming language, C [100] represents the best choice. C is a relatively "low–level" language, and it is suitable for small projects where efficiency and performance are important.

The algorithms for the simulation of multi-volume systems can be easily parallelised because most of the operations are executed considering only the internal state of each volume, hence, a systems composed of n volumes can be simulated by means of n different (parallel) processes which communicate during each iterative cycle to identify the time increment which is used to select the set of rules and, in the final step of the algorithm, during the update operations, to send objects to other volumes and to (possibly) receive objects due to the execution of communication rules within other volumes.

The coding of the τ-DPP and Sτ-DPP has been realised exploiting the message passing interface (MPI) [142]. MPI represents a standard for communication among processes that model a parallel program running on a distributed memory system. Among others, actual distributed memory supercomputers, such as clusters, are suitable for the execution of these programs.

4.7 Discussion and future developments

In this chapter we have shown how the stochastic method based on the tau-leaping procedure can be implemented within the framework of P systems, for the simulation of complex biological systems. In particular, we have considered the class of dynamical probabilistic P systems, to exploit the possibility of modelling systems composed by several volumes.

The new τ selection procedure here introduced works by selecting the smallest τ taken from the set of τs generated inside the membranes during the current iteration; then, an evolution step is performed executing several rules, which are selected following the procedure presented in Section 4.1.

The advantage of introducing the tau-leaping method inside DPPs is that we can choose the same leap of length τ for all the volumes in order to communicate objects in the right way (assuming that they are sent to the other volumes just at the end of each step, because the execution order does not matter); this strategy results in a synchronisation of the volumes and allows to obtain a good approximation of the behaviour of the entire system.

An important aspect is that we can trace the simulated time of the whole system, since every membrane evolves according to the chosen common τ value. Moreover, the time needed to run the simulation with the new procedure is the same as the original tau-leaping algorithm [26] for a single volume. Increasing the number of volumes, the computational time required increases linearly; however, it is very easy to run a simulation on a parallel architecture, simulating one compartment as one process.

In the second part of this chapter we presented a new version of τ-DPP, called Sτ-DPP. The novel properties of Sτ-DPP consist in the representation of the membranes structure and the communication within the system's volumes using two distinct directed graphs, the possibility to define tissue–like structure where nodes have a complex internal architecture, the association of a size to objects and membranes and the consequent handling of the free space during the system evolution.

This new features are suitable for the modelling of a number of real systems in which, first of all, the unlimited accumulation of objects within membranes is not possible or, in other words, where the free space within regions is a critical resource for the system dynamics. Second, the use of two

distinct graphs to describe the membranes structure and the communication within the system provides a formalism with a strong expressive power: indeed it is possible to have communication channels between membranes that are non adjacent and, conversely, it is possible that adjacent membranes do not communicate. The first possibility allows the creation of preferential paths of communication, and can be used, for instance, to reproduce the role of microtubules in the protein transport within cells.

The approaches proposed in this chapter open several interesting research lines, ranging from the modelling of real cellular processes or complex biological systems in general, to the algorithmic improvements of the procedure, and the development of other relevant (modelling and simulating) features in the area of Membrane Computing.

As a possible future development, we can better characterise and study the role of space occupation and diffusion of molecules among the volumes of the modelled systems. Furthermore, the simulation algorithm can be optimised in order to obtain a more efficient procedure and, otherwise, alternative strategies to select the reactions and to handle the reaction applicability can be tested. For instance, we believe that more accurate simulations might be achieved by considering the right–hand sides of communication rules, during the computation of the time step τ, within the target volumes.

In addition, the problem of the size of volumes changing in time should also be addressed, especially when dealing with cellular processes spanning the cell cycle period. In fact, in this case the dimensions of volume can increase, and also its internal conditions can be modified, therefore the requisite condition of fixed volume for the applicability of standard stochastic algorithms fails.

Finally, another possible extension is represented by the consideration of the membrane potential that influences the dynamics of some biological systems [45, 62]. We presented a first study about this topic in [141], where a model for the simulation of the action potential in nervous systems, has been described. Moreover, the τ-DPP algorithm has been modified in order to consider, during the computation of the propensity functions, the influence of the membrane potential on the probability to fire a reaction.

Chapter 5

Modelling chemical and biological systems

Cellular processes are characterized by peculiar features which make them both fascinating and challenging to be studied, either with experimental or computational approaches. To the aim of understanding the complex topology of a biological system and its dynamics, mathematical modeling has extensively proven to represent an indispensable tool [128, 198]. Mathematical modeling can be broadly classified into deterministic continuous approaches, based on the laws of mass action, and into stochastic discrete approaches, which take into consideration the discrete character of the biochemical components and the intrinsic randomness of biological phenomena. Indeed, biological systems can involve a huge number of processes, but the amount of many molecular species can be very low. Under these conditions, deterministic approaches are pushed to their limits and cannot always be relied on, while stochastic methods are able to account for the randomness that emerges and dominates the global behaviour of the system, thus inducing substantial cell–cell phenotypic variations and cellular heterogeneity [59, 111].

In this chapter, different applications of τ-DPP related to the modelling, simulation and analysis of biological and chemical systems will be presented.

In Section 5.1, we present a discrete mathematical model for the Ras/-cAMP/PKA pathway in the yeast *Saccharomyces cerevisiae*, which is involved in the regulation of metabolism and cell cycle progression. The pathway is tightly regulated by several control mechanisms, acting through feedback cycles on the proteins that initiate the cascade events of signal transduction. This system is investigated under various conditions, in order to test how different values of several stochastic reaction constants affect the pathway behaviour. In particular, we show that the level of two elements of this pathway, the guanine nucleotides GTP and GDP, could be relevant metabolic signals for the regulation of the whole system.

In Section 5.2, a model of a genetic oscillator coupled with a quorum sensing intercellular mechanism [58, 67] will be considered. The results of simulations will be presented; some further issues concerning future extensions of the model, as well as different strategies to work out the problems of modelling a genetic oscillator, will be discussed.

In Section 5.3 we introduce the framework of chemical computing, in order to show how to describe computations by means of chemical reaction systems. We will present the encoding of simple boolean functions, of the Fredkin gate and of Fredkin circuits, we also propose an encoding for register machines instructions, and we give some insights about the construction of a complete register machine with n registers. Finally, we will discuss some improvements of this work, with the final aim of arriving at a possible wet implementation of τ–DPP using the micro reactors technology.

Finally, in Section 5.4, the modelling and stochastic simulations of the chemotactic signal transduction pathway in bacteria, are presented. This particular pathway allows bacteria to respond and adapt to environmental changes, by tuning the tumbling and running motions that are due to clockwise and counterclockwise rotations of their flagella.

By exploiting τ–DPP, we analyse this system to consider the interplay between the stochastic fluctuations of a pivotal protein amount and the number of cellular flagella. This approach suggests that the combination of these factors might represent a relevant component for this pathway. Some issues for future extension of this work are finally discussed.

5.1 The Ras/cAMP/PKA signalling pathway in the yeast *Saccharomyces cerevisiae*

In this section, we propose the application of the τ-DPP algorithm for investigating some regulatory mechanisms in the Ras/cAMP/PKA signal transduction pathway in yeast. The outcome of this work gives evidence that τ-DPP represents an efficient strategy to perform stochastic simulations of noisy biological systems. Indeed, a large–scale study has been recently performed to measure the relationship between noise and protein abundance in *Saccharomyces cerevisiae* [147], suggesting that the "intrinsic noise" can dominate the noise in gene expression. Other similar works have corroborated the experimental evidence of noise at single–cell level in yeast, as well as in other microorganisms (reviewed in [111]).

In *S. cerevisiae*, the intracellular signalling molecule (second messenger) cyclic adenosine monophosphate (cAMP) is synthesized by the adenylate cyclase protein, and induces the activation of the cAMP–dependent protein kinase A (PKA). In turn, PKA is able to phosphorylate a variety of proteins involved in transcription, energy metabolism and cell cycle progression ([194, 144]). The Ras/cAMP/PKA pathway plays an important role in the control of yeast cell metabolism, stress resistance and proliferation, in relation to the available nutrients ([193, 194]) and the comprehension of the mechanisms that control this pathway is a relevant question either for basic science or for biotechnological application. The whole signalling cascade is tightly regulated, and the complex interplay between cascade components naturally determines the challenging aspects of the (computational and experimental) investigations on this pathway. Here, we present some results obtained from stochastic simulations of the Ras/cAMP/PKA signalling pathway in a single yeast cell, proving that the model we propose is able to simulate properly the Ras protein cycle, the activation of adenylate cyclase, the synthesis of cAMP, the activation of cAMP–dependent protein kinase PKA and the main feedback. The results have been compared with the experimental data ([175, 38]), and provide information on the key regulatory elements of this signalling network.

5.1.1 The Ras/cAMP/PKA pathway description

In yeast, Ras and cAMP signalling coordinates cell growth and proliferation with nutritional sensing (reviewed in [195]). During the exponential growth phase, the cAMP/PKA pathway downregulates glycogen and trehalose content, stress tolerance, cell–wall resistance to lyticase digestion, and expression of the genes that are controlled by STRE–boxes in their promoters. Moreover, the cAMP/PKA pathway is involved in the control of cell-cycle progression at phases G_1 or G_0, with a modulation of the critical cell size, required both for budding and for mitosis [12]. Different compo-

nents of the Ras/cAMP/PKA signalling pathway can be affected by stress, and contribute to the control of the cellular response. The feeding of glucose (or a related fermentable sugar) to starved *S. cerevisiae* cells, or the induction of intracellular acidification, increase the activity of adenylate cyclase protein, which then triggers a rapid and transient increase in cAMP levels [193].

In yeast, this pathway is tightly regulated (Figure 5.1). The synthesis of cAMP by adenylate cyclase, CYR1, requires the activation of the small GTPases Ras1 and Ras2, two monomeric G proteins which are inactive in the GDP–bound state and active when GTP is bound. Ras proteins are positively controlled by the guanine nucleotide exchange factor (GEFs), Cdc25 and Sdc25, which stimulate the GDP-GTP exchange on Ras, and negatively regulated by the two GTPase activating proteins (GAPs), Ira1 and Ira2, which promote the intrinsically low Ras GTPase activity. Additionally, adenylate cyclase activity is also positively modulated by the heterotrimeric GTPase subunit Gpa2 and by the cognate receptor Gpr1, in response to glucose addition ([37, 175]) (see Figure 5.1).

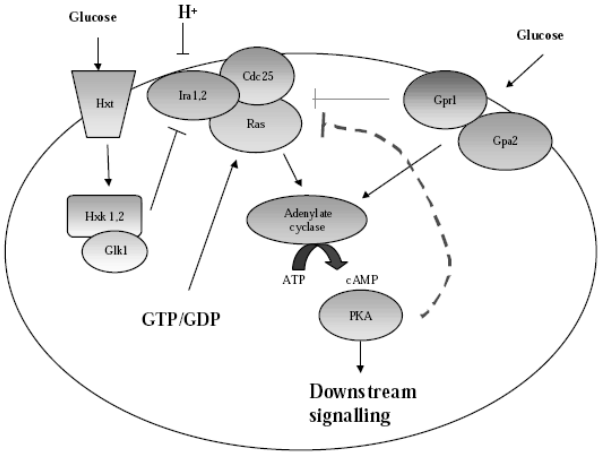


Figure 5.1: The cAMP pathway in yeast.

The protein kinase A (PKA) is the regulatory target of cAMP, which activates the two catalytic subunits of PKA encoded redundantly by TPK1, TPK2 and TPK3 genes ([197, 144]) by binding to its two regulatory subunits (encoded by BCY1 gene). The active PKA operates in downstream intracellular signalling, through the phosphorylation of a variety of proteins involved in transcription, energy metabolism and cell cycle progression ([194, 144]) (Figure 5.1).

The inactivation of cAMP (i.e., the conversion to the form AMP) is governed by the phosphodiesterases Pde1 and Pde2, two enzymes which act as antagonists in signalling in yeast and which constitute the major feedback

control in the Ras/cAMP/PKA pathway ([112, 180]). This strong mechanism operates to rapidly downregulate the pathway, and *S. cerevisiae* mutant cells in which this feedback is inactivated can accumulate large amounts of cAMP ([150]). The cAMP phosphodiesterase Pde1, that is activated by PKA via phosphorylation ([112]), is one of the known target of this feedback, though there are indications that the feedback operates at multiple levels, also modulating the activity of Cdc25 ([81]), Ras or Ira proteins ([38]) in a not clarified manner.

The experimental studies on the Ras/cAMP/PKA pathway are complicated by the fact that both Ras and Cdc25 proteins are not only required for glucose activation of cAMP synthesis, but they are also essential for basal adenylate cyclase activity and for cell viability. Therefore, suppressing mutations, or multicopy suppressor genes restoring viability, have to be present in cells deleted for Ras or Cdc25. These suppressors might affect cAMP accumulation by causing enhanced feedback inhibition or by other unknown means, causing the occurrence of some contradictory results. Recently, Cdc25 has been shown to be essential for Ras2 activation upon glucose addition ([38]) and consequent PKA activation. However, the signals that activate Cdc25 remain elusive, although there are some indications that the level of guanine nucleotides (GTP and GDP) could be a relevant input signal for this pathway ([177]).

5.1.2 The stochastic model

The model represents the Ras/cAMP/PKA pathway in a single yeast cell, it consists of 30 molecular species and is described by the 34 reactions of Table 5.1, where the following notation is used to describe biochemical reactions: X^p means that the molecular species X is phosphorylated, $X + Y$ denotes an interaction between species X and Y, while $X \bullet Y$ denotes that species X and Y are chemically bound in the formation of a complex.

In order to decrease the total number of molecular species, and hence to lower the computational time, we consider only one Ras protein, Ras2 (the most abundant in yeast), and only one GAP, Ira2. The regulatory and catalytic subunits of PKA are denoted by symbols R and C, respectively.

Four main logical modules can be identified in the model:

(1) The switch cycle of Ras2 protein, involving the GEF Cdc25 and the GAP Ira2, described by reactions $r_1, \ldots, r_{10}$.

$$\begin{array}{ccccc} Ras2 \bullet GDP + Cdc25 & \rightleftarrows & Ras2 \bullet GDP \bullet Cdc25 & \rightleftarrows & Ras2 \bullet Cdc25 + GDP \\ & & & & \downarrow\uparrow \\ Ras2 \bullet GTP + Cdc25 & \rightleftarrows & Ras2 \bullet GTP \bullet Cdc25 & \rightleftarrows & Ras2 \bullet Cdc25 + GTP \end{array}$$

The nucleotide exchange cycle has been derived by using the data provided in [84, 104, 177], which show the existence of a nucleotide free $Ras \bullet Cdc25$ complex that can generate either $Ras \bullet GDP$ or

$Ras \bullet GTP$ with similar kinetics, suggesting that the "exchange" is dependent on the concentration of available nucleotides and is not "directional" toward the generation of $Ras \bullet GTP$.

(2) The synthesis of cAMP, via activation of adenylate cyclase CYR1, described by reactions $r_{11}, \ldots, r_{13}$. This is modeled assuming that adenylate cyclase (CYR1) is active only in a complex with $Ras2 \bullet GTP$. The reaction r_{13} is optional and include a cyclase–dependent interaction of Ras2 with Ira, as suggested by some experiments [119].

(3) The activation of PKA, via the reversible binding between its regulatory subunits R and cAMP, and the subsequent dissociation of the tetramer PKA, described by reactions $r_{14}, \ldots, r_{24}$.

(4) The activity of phosphodiesterases Pde1 and Pde2 (the latter is active in the basal level regulation of cAMP), which determines the feedback mechanisms for cAMP degradation, described by reactions $r_{25}, \ldots, r_{32}$. A Michaelis–Menten kinetics is simulated for Pde1 and Pde2. A phosphatase activity (PPA2) is also introduced to reset the system. In addition, reactions r_{33} and r_{34} include an additional feedback mechanism based on the phosphorylation/dephosphorylation of Cdc25, as suggested in [81].

The number of molecules for Ras2, Cdc25, PKA, Pde1, Pde2 and PPA2 was obtained by using the data of [70] (available online at [205]).

The number for Ira2 and CYR1 was estimated by comparing the fluorescence of yeast cells expressing fusion with eGFP (obtained by [205]) using Cdc25–eGFP as a standard (300 molecules/cell).

The number of ATP, GTP and GDP was calculated by data of [177], considering an average cell volume of $45fL$ [27, 96]. Taking into account that part of the volume is given by cell wall and by internal structures, we estimated an internal free water volume V of about $30fL$ ($3 \times 10^{-14}L$).

Therefore, for ATP (considering a concentration of $1mM$) we obtained about 2×10^7 molecules/cell, while for GTP and GDP we estimated 5×10^6 and 1.5×10^6 molecules/cell, respectively, for yeast cells growing in minimal glucose medium. In a similar way we calculated the number of cAMP molecules using either literature [175] or experimental data.

The amount of different molecular species, that is, the discrete number of molecules per cell, are listed in Table 5.2.

The estimation of the stochastic reaction constants of the model was achieved by testing both the effect of a range of values for each constant within any module, and the response of each and every module. More specifically, starting from the simulation of the Ras2 switching cycle (module 1), we assumed plausible relative magnitudes of the stochastic constants for reactions $r_1, \ldots, r_{10}$, and we adjusted them one by one, till we obtained a

Table 5.1: The model for the Ras/cAMP/PKA pathway consisting of 34 reactions, given in the form "reagents → products", and the corresponding stochastic reaction constants.

Reaction	Reagents	Products	Constant
r_1	$Ras2 \bullet GDP + Cdc25$	$Ras2 \bullet GDP \bullet Cdc25$	1.0
r_2	$Ras2 \bullet GDP \bullet Cdc25$	$Ras2 \bullet GDP + Cdc25$	1.0
r_3	$Ras2 \bullet GDP \bullet Cdc25$	$Ras2 \bullet Cdc25 + GDP$	1.5
r_4	$Ras2 \bullet Cdc25 + GDP$	$Ras2 \bullet GDP \bullet Cdc25$	1.0
r_5	$Ras2 \bullet Cdc25 + GTP$	$Ras2 \bullet GTP \bullet Cdc25$	1.0
r_6	$Ras2 \bullet GTP \bullet Cdc25$	$Ras2 \bullet Cdc25 + GTP$	1.0
r_7	$Ras2 \bullet GTP \bullet Cdc25$	$Ras2 \bullet GTP + Cdc25$	1.0
r_8	$Ras2 \bullet GTP + Cdc25$	$Ras2 \bullet GTP \bullet Cdc25$	1.0
r_9	$Ras2 \bullet GTP + Ira2$	$Ras2 \bullet GTP \bullet Ira2$	3.0×10^{-2}
r_{10}	$Ras2 \bullet GTP \bullet Ira2$	$Ras2 \bullet GDP + Ira2$	7.0×10^{-1}
r_{11}	$Ras2 \bullet GTP + CYR1$	$Ras2 \bullet GTP \bullet CYR1$	1.0×10^{-3}
r_{12}	$Ras2 \bullet GTP \bullet CYR1 + ATP$	$Ras2 \bullet GTP \bullet CYR1 + cAMP$	2.1×10^{-6}
r_{13}	$Ras2 \bullet GTP \bullet CYR1 + Ira2$	$Ras2 \bullet GDP + CYR1 + Ira2$	1.0×10^{-3}
r_{14}	$cAMP + PKA$	$cAMP \bullet PKA$	1.0×10^{-5}
r_{15}	$cAMP + cAMP \bullet PKA$	$(2cAMP) \bullet PKA$	1.0×10^{-5}
r_{16}	$cAMP + (2cAMP) \bullet PKA$	$(3cAMP) \bullet PKA$	1.0×10^{-5}
r_{17}	$cAMP + (3cAMP) \bullet PKA$	$(4cAMP) \bullet PKA$	1.0×10^{-5}
r_{18}	$(4cAMP) \bullet PKA$	$cAMP + (3cAMP) \bullet PKA$	1.0×10^{-1}
r_{19}	$(3cAMP) \bullet PKA$	$cAMP + (2cAMP) \bullet PKA$	1.0×10^{-1}
r_{20}	$(2cAMP) \bullet PKA$	$cAMP + cAMP \bullet PKA$	1.0×10^{-1}
r_{21}	$cAMP \bullet PKA$	$cAMP + PKA$	1.0×10^{-1}
r_{22}	$(4cAMP) \bullet PKA$	$2C + 2(R \bullet 2cAMP)$	1.0
r_{23}	$R \bullet 2cAMP$	$R + 2cAMP$	1.0
r_{24}	$2R + 2C$	PKA	1.0
r_{25}	$C + Pde1$	$C + Pde1^p$	1.0×10^{-6}
r_{26}	$cAMP + Pde1^p$	$cAMP \bullet Pde1^p$	1.0×10^{-1}
r_{27}	$cAMP \bullet Pde1^p$	$cAMP + Pde1^p$	1.0×10^{-1}
r_{28}	$cAMP \bullet Pde1^p$	$AMP + Pde1^p$	7.5
r_{29}	$Pde1^p + PPA2$	$Pde1 + PPA2$	1.0×10^{-4}
r_{30}	$cAMP + Pde2$	$cAMP \bullet Pde2$	1.0×10^{-4}
r_{31}	$cAMP \bullet Pde2$	$cAMP + Pde2$	1.0
r_{32}	$cAMP \bullet Pde2$	$AMP + Pde2$	1.7
r_{33}	$C + Cdc25$	$C + Cdc25^p$	10.0
r_{34}	$Cdc25^p + PPA2$	$Cdc25 + PPA2$	1.0×10^{-2}

good reproduction of the expected behaviour of the subsystem described by this module. Then, every other module has been sequentially added to the first one, following the same iterative process, and tested together to finally perform a comprehensive and correct simulation of the whole pathway. Our set of derived stochastic constants is listed in Table 5.1; the values correspond to those used in the simulations of the following sections, when not otherwise specified. In Table 5.1, all stochastic constants are expressed in arbitrary units (time^{-1}).

For the simulations of the model we used the τ-DPP stochastic algorithm, described in Chapter 4. In our simulations, the value of error control parameter ϵ (which is related to the accuracy level of the algorithm) has been set to 0.03, as suggested in [26].

Table 5.2: The copy number for Cdc25, Pde1, PKA, PPA2, Pde2 and Ras2 were taken by [70], we assumed that at the beginning of simulations all Ras2 is associated with GDP (Ras2 • GDP); the number of CYR1 and Ira2 was estimated by the fluorescence of GFP fusions; the number of molecule per cell of GDP, GTP and ATP were derived by data reported in [177].

Molecular species	Copy numbers
$CYR1$	200
$Cdc25$	300
$Ira2$	200
$Pde1$	1400
PKA	2500
$PPA2$	4000
$Pde2$	6500
$Ras2 \bullet GDP$	20000
GDP	1.5×10^6
GTP	5.0×10^6
ATP	2.4×10^7

5.1.3 The Ras2 • GTP generation module

The starting point of our stochastic simulations is the module corresponding to the switching (activation/inactivation) cycle of protein Ras2, that is, reactions $r_1, \ldots, r_{10}$ in Table 5.1. The generation of the complex Ras2 • GTP turns out to be very sensitive to the concentration of guanine nucleotides (GTP).

The system starts with only Ras2 • GDP complex (20000 molecules), and with GTP and GDP amounts equal to 5×10^6 and 1.5×10^6 molecules, respectively. This situation is comparable with that found during yeast growth on minimal media, where the Ras2 • GTP level increases to a value corresponding to 4% of the total Ras2 value [177, 38]. When the GTP level is decreased from 5×10^6 to 1.5×10^6 - a level that mimics a situation of starved cells [177] - the amount of Ras2 • GTP drops to a very low level (less than 0.5%) (Figure 5.2).

As expected, the Ras2 • GTP complex is greatly influenced by the rate of dissociation of Cdc25 (stochastic constant c_7) and by the activity of Ira2 (stochastic constant c_{10}), as shown in Figure 5.3.

The remaining stochastic reaction constants of this module give only a lower contribution, as summarized in Figure 5.4, where we show the parametric sensitivity of the free Ras2 • GTP level. The variation of the stochastic constants c_4, c_5, c_6, c_9 did not result in significant Ras2 • GTP response, hence we report only the outcome due to the changes in c_1, c_2, c_3, c_7, c_8,

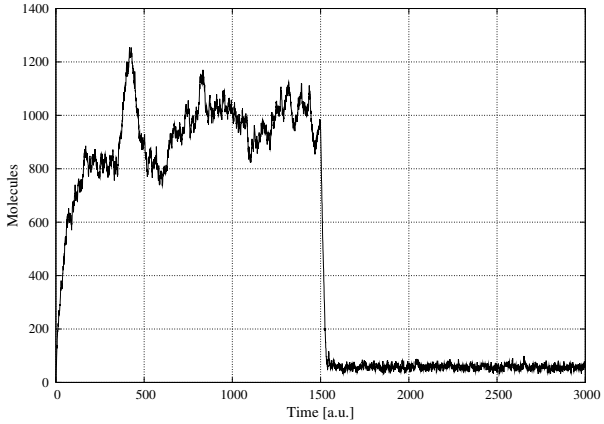


Figure 5.2: Variation of Ras2 • GTP dependent on a step decrease of GTP amount (from 5×10^6 to 1.5×10^6 molecules) taking place at time 1500.

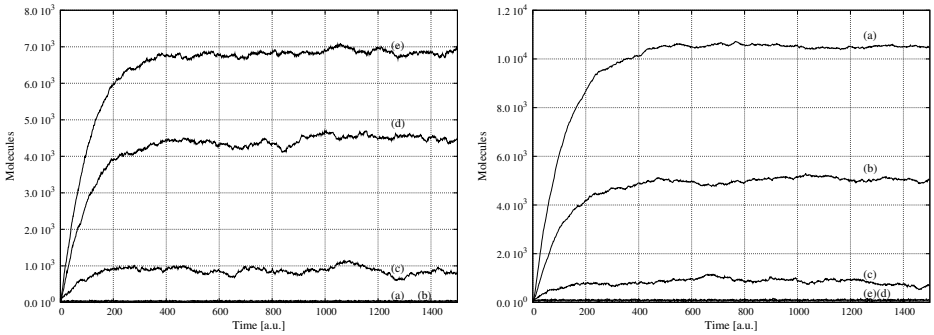


Figure 5.3: Sensitivity of Ras2 • GTP. (Left figure) Dependence on the rate of dissociation of Cdc25: (a) $c_7 = 0.2$; (b) $c_7 = 0.5$; (c) $c_7 = 1.0$; (d) $c_7 = 1.5$ and (e) $c_7 = 2.0$.(Right figure) Dependence on the activity of Ira2: (a) $c_{10} = 0.25$; (b) $c_{10} = 0.5$; (c) $c_{10} = 0.7$; (d) $c_{10} = 0.9$ and (e) $c_{10} = 1.1$.

c_{10} values.

5.1.4 Generation of cAMP, coupling with PKA and feedback loops

The coupling of Ras2 cycle (reactions $r_1, \ldots, r_{10}$) with adenylate cyclase activity (reactions r_{11}, r_{12}, r_{13}) allows the accumulation of cAMP in response to Ras2 • GTP. On the other hand, the reactions r_{30}, r_{31} and r_{32} cause a degradation of cAMP to AMP by the cAMP–high–affinity phosphodiesterase, Pde2 (a Michaelis–Menten kinetics is used to describe this interaction). This allows the accumulation of cAMP (Figure 5.5) to very high levels, comparable to those experimentally observed in mutant yeast cells where the feedback mechanism is attenuated or inactivated (that is, in cells

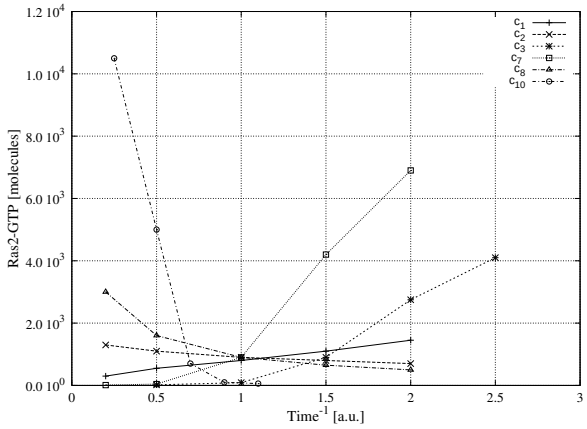


Figure 5.4: Sensitivity of Ras2 • GTP module.

where phosphodiesterase Pde1 is not fully working) [150]. In this simulation, in order to switch off the activity of Pde1, we set to zero the values of rule constants $c_{25}, \ldots, c_{29}$. As expected, the steady–state cAMP level depends upon the activity of Pde2 (Figure 5.5).

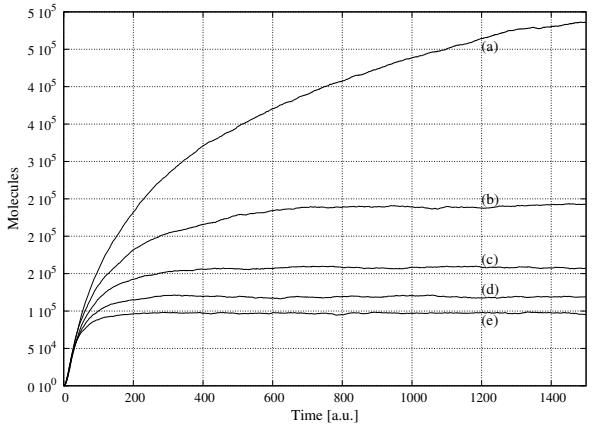


Figure 5.5: Effect of Pde2 activity on cAMP accumulation: (a) $c_{32} = 1.6$; (b) $c_{32} = 1.7$; (c) $c_{32} = 1.8$; (d) $c_{32} = 1.9$ and (e) $c_{32} = 2.0$.

If we add the basic feedback mechanism (reactions $r_{25}, \ldots, r_{29}$) based on the activity of the cAMP low affinity phosphodiesterase Pde1 [112], we obtain a transient accumulation of cAMP (Figure 5.6). Moreover, the catalytic subunits of PKA are clearly modulated by the level of cAMP (data not shown).

In Figure 5.7 we also show how the variation of affinity between cAMP and PKA modifies the PKA activity, as well as the cAMP level (reactions $r_{14}, \ldots, r_{24}$).

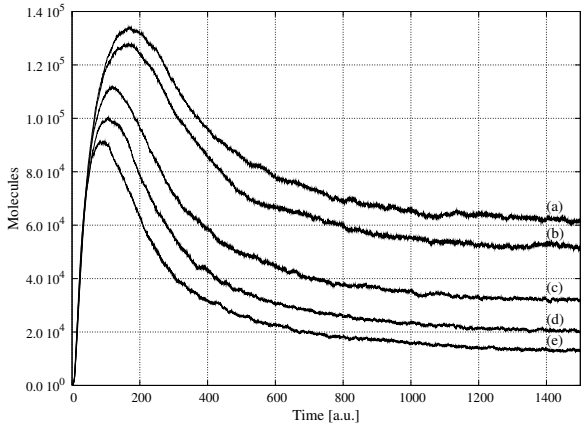


Figure 5.6: Effect of Pde1 activity on cAMP accumulation: (a) $c_{28} = 1.7$; (b) $c_{28} = 2.0$; (c) $c_{28} = 3.0$; (d) $c_{28} = 4.0$ and (e) $c_{28} = 5.0$.

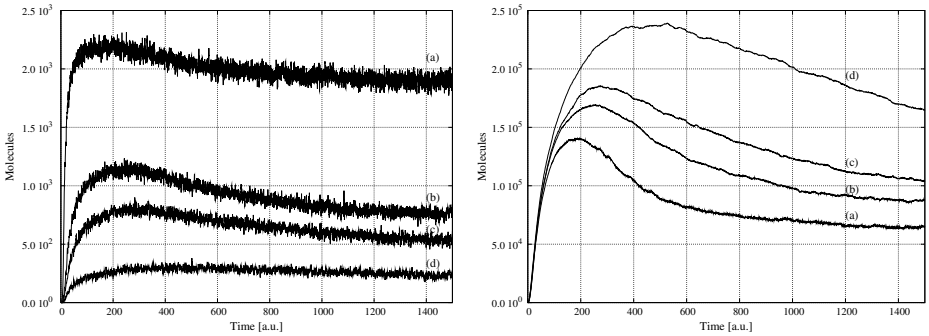


Figure 5.7: Variation of the catalytic subunit of PKA (left) and cAMP (right) dependent on the affinity between PKA and cAMP (rules $r_{14}, \ldots, r_{21}$). Values of reaction constants: (a) $c_i = 10^{-4}$, $c_j = 0.1$; (b) $c_i = 10^{-5}$, $c_j = 0.1$; (c) $c_i = 5 \times 10^{-6}$, $c_j = 0.1$ and (d) $c_i = 10^{-6}$, $c_j = 0.1$, where $i = 14, 15, 16, 17$; $j = 18, 19, 20, 21$.

We then simulate a transition in GTP amount by means of one step increase, which mimics the glucose induced increase of intracellular GTP [177]. As shown in Figure 5.8, a response close to the one observed in vivo is achieved (for a comparison see Figure 2A of [175]).

This can be considered as an additional proof that our model simulates in a reasonable quantitative way the cAMP pathway in a single yeast cell.

Finally, in Figure 5.9 we show how different values of GTP amount affect both the peak and the basal level of cAMP (left), and PKA activity and Ras2 • GTP level (right). Hence, we can demonstrate the strong dependence of the signalling pathway on GTP levels, indicating that the intracellular GTP/GDP ratio, that is modulated by nutrients availability,

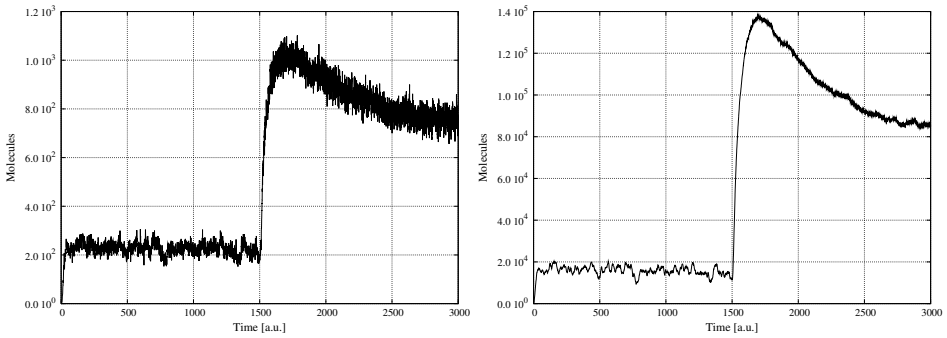


Figure 5.8: Variation of C (left) and of cAMP (right) dependent on a step increase of GTP amount (from 1.5×10^6 to 5×10^6 molecules) taking place at time 1500.

may be a relevant signal for this pathway as also suggested in [84, 177].

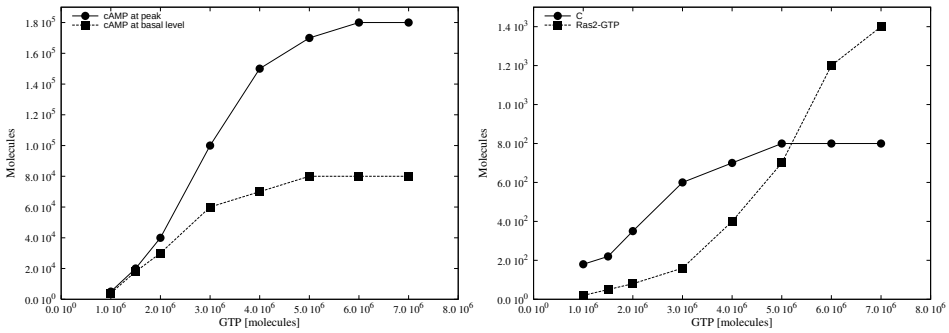


Figure 5.9: Effect of different GTP input values on cAMP accumulation (left) and on Ras2 • GTP and PKA activity (right).

5.1.5 Discussion

In mathematical modeling of biological or biochemical systems, sensitivity analysis is used to quantify how the parameters of the model can affect the system behaviour, e.g., to help in recognizing the fragilities or the robustness of the system. Though it has been traditionally applied to deterministic continuous models, theories and tools for parametric sensitivity of discrete stochastic systems have recently been defined [82, 166], also with the aim of capturing the relevant stochastic effects which can occur in small systems. Through the simulations presented before, we determined - by manually changing the values one by one - how small variations in some stochastic constants of the Ras2 switching cycle (the first module of our model) result in relative high variations of Ras2 • GTP level in the yeast cell (Figure 5.4). Further analysis of this type, via appropriate algorithmic methods, is an

ongoing work and will be elaborated for the whole pathway.

Indeed, a crucial point in understanding the dynamics of biological systems through computer simulations is related to the evaluation of the parameters involved in the system, such as molecular concentrations (or copy numbers), binding constants, transcription or translation rates, etc. Except for few special cases where the values can be found in literature, these constants are not available or else ambiguous. The lack of information and the inaccuracy about these data are often due to the difficulty, or the impossibility, to perform correct measures during in vivo or in vitro experiments.

This fact then results in the challenging problem of assigning a reasonable temporal dimension to the (stochastic) simulation results. The problem of calibrating the system's parameters can be tackled by using optimisation techniques in order to find a set of optimal parameter values which can be used to reproduce the observed dynamics of a system, as explained in Chapter 6.

For the simulations of the Ras/cAMP/PKA pathway, we were able to make a direct comparison between the simulations and the experimental results. In yeast, cAMP was experimentally found to be around 2×10^5 molecules after stimulation, while basal levels were around 2-5 $\times 10^4$ molecules/cell. In addition, after stimulation, a cAMP peak was observed after 45-60 s, then a decrease was observed and a new steady–state reached in 3-5 min (for a direct comparison see the paper [175], Figure 2). These experimental data fit very well with our stochastic simulations, hence, we are in the condition to assign a reasonable temporal scale to our simulations. A similar quantitative comparison can be done for Ras2 • GTP, which was experimentally found to be in the range of 2-4% of total Ras2 in growing cells (equivalent to 400-800 molecules/cell).

In addition, after stimulation of starved cells with glucose, Ras2 • GTP increased from 0.5 to 4% of total Ras2 in about 1 min [177], and again these data fit very well with our simulations.

In this work, we focused on the cytoplasmic regulators of the Ras/-cAMP/PKA pathway, and provided stochastic simulations of the synthesis of cAMP and of the cytoplasmic quantities of the pivotal complexes. As a future development, we will construct an accurate map of related cellular phenomena, such as the response of this pathway to nutrients and to intracellular acidification, its connections with other pathways co–involved in glucose signalling, and the downhill nuclear expression of target genes. Here, we have used the level of intracellular GTP as input, that was experimentally found to quickly respond to nutrients availability [177], but we could also simulate the effect of a decrease of intracellular pH (that likely causes an inhibition of GAP activity of the Ira proteins [37], or a fast increase of Cdc25 activity [38]). In order to gain a higher biological relevance of our model, we will include the Gpr1/Gpa2 pathway, which is a specific signalling mechanism that responds only to high glucose concentration, and operates

together with Ras2 • GTP to activate adenylate cyclase [37, 175]. Finally, we will also test the effect of additional feedback mechanisms, such as the inactivation of Cdc25, the activation of Ira, etc.

We also expect that our computational studies will support the experimental work aimed at characterizing the actual localizations (that is, either bound to plasma membrane or to internal membranes, or diffused within the cell) of the central components of this pathway, since there exists evidence that most of the Cdc25, CYR1, Ira2, Ras2 proteins localize in internal membranes, suggesting the presence of large signalling complexes inside the yeast cells [119].

The τ-DPP algorithm [30, 118] will be exploited in order to investigate the topological distribution of the molecular species in distinct cellular regions, and to introduce additional components and other external inputs.

Finally, we believe that establishing the pivotal roles of the different components in the Ras/cAMP/PKA pathway in yeast might have a positive outcome for the elucidation of similar components in higher eukaryotes. For instance, it is well known that the protein neurofibromin 1 (NF1), acting as tumor suppressor in human cells, is homologous to the proteins of the Ira family, and it has been shown to contain a GAP related domain which regulates Ras [10]. Though its evident importance, biochemical and cellular characterizations of NF1 are still on the way, and very few knowledge is currently available about NF1-interacting proteins. Therefore, the synergistic integration of pathway modeling and of deletion experiments about Ira proteins in yeast could probably provide useful hints for further interdisciplinary investigations of NF1.

5.2 The repressilator: a genetic oscillators coupled with a quorum sensing mechanism

The control of gene expression in living cells can occur through distinct mechanisms, such as positive or negative regulation, which enhances (or inhibits, respectively) the binding between the RNA polymerase and the promoter site of a gene. One goal in gene regulation analysis is to understand how functional oscillations (e.g., in the circadian clock) can emerge in a complex system as the macroscopic effect of the interactions and the coupling of basic and microscopic elements.

In particular, the capability to construct and control *synthetic* gene circuits in laboratory experiments, allows to investigate the issue of gene regulation in simplified systems. An example of such systems is the *repressilator*, a synthetic oscillator implemented in *Escherichia coli* cells by means of specifically constructed plasmids [58]. It consists of a network of three genes, *lacI* (from *E. coli*), *tetR* (from the tetracycline-resistant transposon Tn10), and *cI* (from λ phage), whose codified proteins act as repressors of each other's gene, in a cyclic way. Namely, the product of *lacI* inhibits the transcription of *tetR*, the product of *tetR* inhibits the transcription of *cI*, whose product in turn inhibits the transcription of *lacI*, thus closing the repression cycle. To detect and readout the network behaviour, a green fluorescent protein – whose synthesis is periodically triggered by the *tetR* product – has been used as the reporter part of this system. Observations on a growing *E. coli* culture evidenced the emergence of spontaneous oscillations in individual cells, as well as the effect of noise through the variability between different cells, probably due to the stochastic fluctuations of the network components [58].

Another important aspect of bacteria is that they are able to synthesize a diffusive molecule, called *autoinducer*, which is used to perform intercellular signalling. This communication mechanism, termed *quorum sensing*, allows bacterial cells to sense whether a critical cell density has been reached in their surrounding, thereby switching on whole-population behaviours through the synchronization of all individuals.

Gram-positive and Gram-negative bacteria synthesize different signalling molecules. In general, the first use olipeptides as autoinducers, while the latter (e.g. *E. coli*) use acyl-homoserine lactone (acyl-HSL) molecules. The quorum sensing circuits in Gram-negative bacteria usually contain two main families of proteins, which have been found to be homologous to two well-characterized protein in the bioluminescent bacterium *V. fischeri*: the LuxI and LuxR families. LuxI-like proteins are needed for the synthesis of the autoinducers (they acts as acyl-HSL synthases when provided with an amino donor and an appropriate acyl donor [152]), while LuxR-like proteins form complexes with the autoinducer, which then regulate the transcription of

target genes [132].

In [67], a synthetic multicellular clock has been investigated, by coupling the repressilator system with bacterial quorum sensing. This intercellular communication mechanism is able to lead the *local* genetic oscillators, within a noisy and nonidentical population, to *global* oscillatory rhythms. In particular, it was shown that individual repressilator systems can self-synchronize, even when their periods are broadly distributed [67].

In this section, we propose a multivolume model based on τ-DPP that considers a repressilator-like system (RL) coupled with a quorum sensing-like circuit (QSL). Namely, the RL consists of three genes (G_1, G_2, G_3) which cyclically inhibit each other's expression, while the QSL consists of a genetic component that triggers the production of the autoinducer (through the synthesis of the intermediate autoinducer-synthases), plus a "control" molecule that mimics the role of the protein in the LuxR family. The two systems are interlaced through one *positive feedback* and one *negative feedforward* loop: the first operates on the promotion of gene G_1 in the RL by means of the QSL autoinducer molecule, while the second operates on the inhibition of the QSL gene by means of G_1 product (see Figure 5.10).

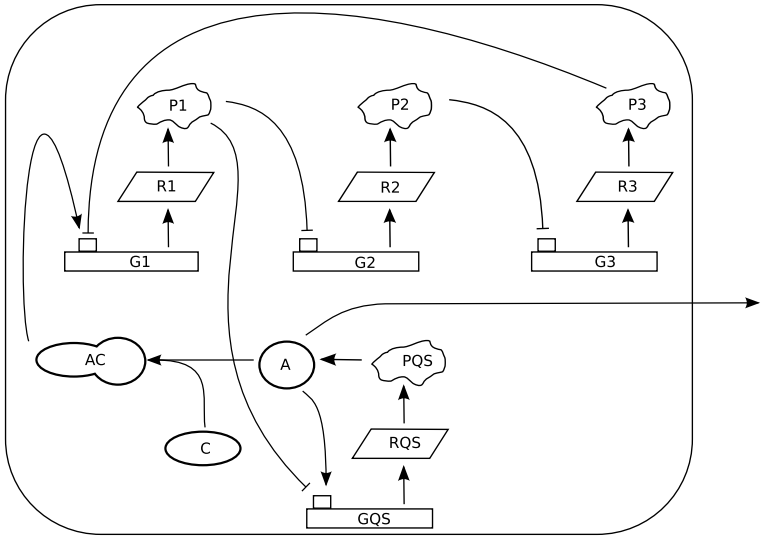


Figure 5.10: The (single volume) network of the genetic oscillator system coupled with quorum sensing mechanism.

5.2.1 A multivolume model for coupled genetic oscillators

The multivolume model for RL + QSL system consists of n volumes, each one corresponding to a cell, a set of 34 reactions defined within each cell (Table 5.3), 1 communication reaction defined in the environment (Table 5.4),

and an alphabet of 21 molecular species. Among these, we can recognize 14 elementary species, and 7 complex species.

The set $\mathcal{S}$ of elementary species consists of G_i, R_i, P_i, which denote, respectively, the i-th gene, mRNA and protein in the RL, for each $i = 1, 2, 3$; G_{QS}, R_{QS}, P_{QS} which denote, respectively, the gene, mRNA and protein in the QLS; A, the autoinducer molecule; C, the control molecule used to trigger on, along with A, the overexpression of gene G_1. The set of complex species is denoted by $\mathcal{C}$, and consists of the species $G_1 \cdot P_3, G_2 \cdot P_1, G_3 \cdot P_2, A \cdot C, A \cdot C \cdot G_1, A \cdot G_{QS}, G_{QS} \cdot P_1$. Each complex species is formed when two elementary species interact and get bound. Formally, this process is represented by means of rules of the form $X + Y \rightarrow X \cdot Y$, where $X, Y \in \mathcal{S}$. We also consider the possibility to have complex species consisting of more than two elementary species; in this case the complex is formed in two sequential steps, that is, $X + Y \rightarrow X \cdot Y$, followed by $X \cdot Y + Z \rightarrow X \cdot Y \cdot Z$, where $X, Y, Z \in \mathcal{S}$.

The rules in each cell work as follows. Rules r_1 and r_2 describe, respectively, the transcription of gene G_1 into one molecule of mRNA, R_1, and its translation into one copy of protein P_1. Rules r_3, r_4 describe the degradation of R_1 and P_1, respectively. The same holds for the sets of rules $r_5, \ldots, r_8$ and $r_9, \ldots, r_{12}$, defined for the other two genetic elements in the RL[1].

Rules r_{13}, r_{15}, r_{17} describe the cyclic repression relation between the 3 genes in the RL: namely, the protein synthesized by gene G_1, G_2, G_3, acts as repressor for the expression of gene G_2, G_3, G_1, respectively. When the protein binds to the gene (e.g., rule r_{13} forms the complex $G_1 \cdot P_3$), it blocks gene expression by avoiding the application of the respective transcription rule (e.g., rule r_1). The gene repression ends whenever the repressor is released from the complex and the gene returns in an unbound form, that is, by the application of the inverse rules r_{14}, r_{16}, r_{18}.

The other rules consider the coupling between the RL and the QSL, which allows cells to communicate each other; as a consequence, their behaviour can get synchronized. Rules $r_{19}, \ldots, r_{22}$ describe the transcription and translation of the quorum sensing gene G_{QS}, and the degradation of the respective synthesized mRNA and protein, R_{QS} and P_{QS}. Rule r_{23} describes the synthesis of the autoinducer molecule A, by means of the quorum sensing protein P_{QS}[2]. When the autoinducer is present in the membrane, it can undergo three different processes. First, it can bind to the quorum sensing gene (rule r_{24}, and its inverse r_{25}) and promote its expression (rule r_{26}); note that the expression of G_{QS} can also be inhibited, if protein P_1

[1]Note that we do not explicitly describe the presence and the role played by transcriptional, translational and degradation machineries occurring in real cells, tacitly assuming their constant availability.

[2]In bacteria, the synthesis of these molecules require the additional components described in Section 5.2. We assume that these components are always available and we do not explicitly include them in the model.

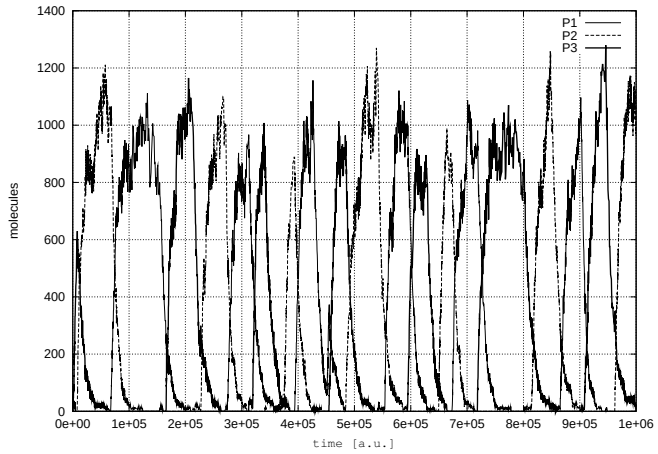


Figure 5.11: Stochastic oscillations of the repressor proteins P_1, P_2, P_3 in the single cell.

binds to it (rule r_{27}, and its inverse r_{28}). Second, the autoinducer can form a complex with the control molecules C (rule r_{29}, and its inverse r_{30}). In this case, the complex $A \cdot C$ can bind to gene G_1 (rules r_{31} and r_{32}) and enhance its expression (rule r_{33}). Third, the autoinducer can exit the cell (rule r_{34}) and diffuse into the environment, from where it can enter again any cell, by the application of the non deterministic environmental rule r_E.

5.2.2 Results of simulations

In the following, we report the results of the simulations performed with τ-DPP using, when not otherwise specified, the stochastic constant values given in Tables 5.3 and 5.4.

We start by presenting in Figure 5.11 the oscillatory dynamics of the repressor proteins P_1, P_2, P_3 in the single cell, when the quorum sensing circuit is silenced (that is, only rules $r_1, \ldots, r_{18}$ are active inside the cell). The expected outcome resembles the behaviour obtained in [58], where the order of oscillations of the three proteins proceeds in a sequential, cyclic manner, dictated by the structure of the genetic repressor systems.

In Figure 5.12 we show the variation of dynamics of the quorum sensing protein, P_{QS}. On the left side, we show the situation where only rules $r_1, \ldots, r_{22}$ are active, hence the RL and the QSL are not interlaced. In this case, P_{QS} reaches a steady state (top part), and the complex $G_{QS} \cdot P_1$ is never formed (bottom part). On the contrary, if also rules r_{27}, r_{28} are active, that is, the negative feedforward occurs, then P_{QS} undergoes controlled oscillations (right side, top part). The right bottom part indicates how frequently protein P_1 regulates the expression of G_{QS} (the complex $G_{QS} \cdot P_1$ is formed).

Table 5.3: Reactions inside the single cell.

Reaction	**Reagents → products**	**Constant**
r_1	$G_1 \to G_1 + R_1$	$c_1 = 1 \cdot 10^{-2}$
r_2	$R_1 \to R_1 + P_1$	$c_2 = 1 \cdot 10^{-1}$
r_3	$R_1 \to \lambda$	$c_3 = 1 \cdot 10^{-4}$
r_4	$P_1 \to \lambda$	$c_4 = 1 \cdot 10^{-2}$
r_5	$G_2 \to G_2 + R_2$	$c_5 = 1 \cdot 10^{-2}$
r_6	$R_2 \to R_2 + P_2$	$c_6 = 1 \cdot 10^{-1}$
r_7	$R_2 \to \lambda$	$c_7 = 1 \cdot 10^{-4}$
r_8	$P_2 \to \lambda$	$c_8 = 1 \cdot 10^{-2}$
r_9	$G_3 \to G_3 + R_3$	$c_9 = 1 \cdot 10^{-2}$
r_{10}	$R_3 \to R_3 + P_3$	$c_{10} = 1 \cdot 10^{-1}$
r_{11}	$R_3 \to \lambda$	$c_{11} = 1 \cdot 10^{-4}$
r_{12}	$P_3 \to \lambda$	$c_{12} = 1 \cdot 10^{-2}$
r_{13}	$G_1 + P_3 \to G_1 \cdot P_3$	$c_{13} = 1 \cdot 10^{-1}$
r_{14}	$G_1 \cdot P_3 \to G_1 + P_3$	$c_{14} = 1 \cdot 10^{-3}$
r_{15}	$G_2 + P_1 \to G_2 \cdot P_1$	$c_{15} = 1 \cdot 10^{-1}$
r_{16}	$G_2 \cdot P_1 \to G_2 + P_1$	$c_{16} = 1 \cdot 10^{-3}$
r_{17}	$G_3 + P_2 \to G_3 \cdot P_2$	$c_{17} = 1 \cdot 10^{-1}$
r_{18}	$G_3 \cdot P_2 \to G_3 + P_2$	$c_{18} = 1 \cdot 10^{-3}$
r_{19}	$G_{QS} \to G_{QS} + R_{QS}$	$c_{19} = 1 \cdot 10^{-2}$
r_{20}	$R_{QS} \to R_{QS} + P_{QS}$	$c_{20} = 1 \cdot 10^{-1}$
r_{21}	$R_{QS} \to \lambda$	$c_{21} = 1 \cdot 10^{-4}$
r_{22}	$P_{QS} \to \lambda$	$c_{22} = 1 \cdot 10^{-2}$
r_{23}	$P_{QS} \to P_{QS} + A$	$c_{23} = 5 \cdot 10^{-3}$
r_{24}	$G_{QS} + A \to G_{QS} \cdot A$	$c_{24} = 1 \cdot 10^{-7}$
r_{25}	$G_{QS} \cdot A \to G_{QS} + A$	$c_{25} = 1 \cdot 10^{-3}$
r_{26}	$G_{QS} \cdot A \to G_{QS} \cdot A + R_{QS}$	$c_{26} = 3 \cdot 10^{-2}$
r_{27}	$G_{QS} + P_1 \to G_{QS} \cdot P_1$	$c_{27} = 1 \cdot 10^{-2}$
r_{28}	$G_{QS} \cdot P_1 \to G_{QS} + P_1$	$c_{28} = 1 \cdot 10^{-3}$
r_{29}	$A + C \to A \cdot C$	$c_{29} = 1 \cdot 10^{-3}$
r_{30}	$A \cdot C \to A + C$	$c_{30} = 1 \cdot 10^{-2}$
r_{31}	$G_1 + A \cdot C \to G_1 \cdot A \cdot C$	$c_{31} = 1 \cdot 10^{-6}$
r_{32}	$G_1 \cdot A \cdot C \to G_1 + A \cdot C$	$c_{32} = 1 \cdot 10^{-3}$
r_{33}	$G_1 \cdot A \cdot C \to G_1 \cdot A \cdot C + R_1$	$c_{33} = 5 \cdot 10^{-2}$
r_{34}	$A \to (A, out)$	$c_{34} = 1$

Table 5.4: Reaction in the environment.

Reaction	**Reagents → products**	**Constant**
r_E	$A \to (A, in)$	$c_E = 1$

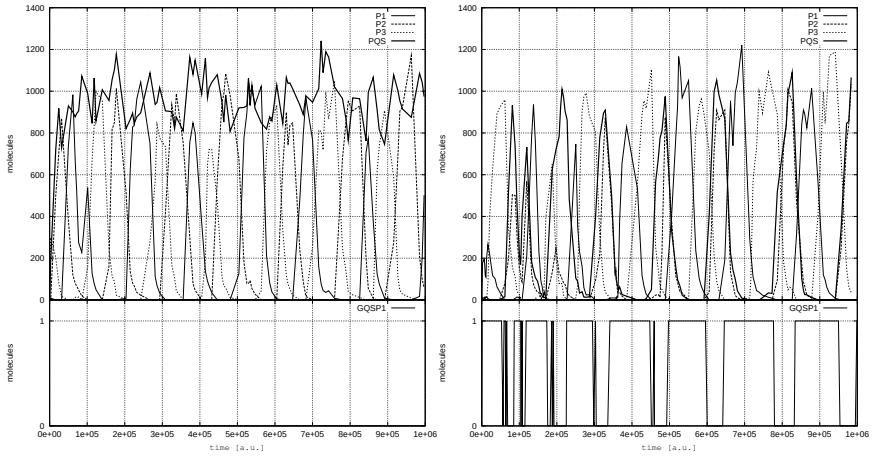


Figure 5.12: Dynamics of quorum sensing protein P_{QS}. Left side: steady state. Right side: controlled oscillations.

Then, we consider the communication between the single cell and the environment, and investigate the oscillatory dynamics within the cell through the activation, from time to time, of different subsets of rules.

In Figure 5.13 we show the situation within the single cell when the quorum sensing circuit is activated and interlaced with the genetic oscillator. On the left side, all rules $r_1, \ldots, r_{26}, r_{34}, r_E$ are active, but the control of P_1 over G_{QS} is silenced. In this case, P_{QS} reaches again a steady state (top part), and the autoinducer molecule is constantly accumulated inside the cell (bottom part), thus promoting the expression of G_{QS} though the formation of the complex $G_{QS} \cdot A$ (middle part). If also rules r_{27}, r_{28} are activated (right side), then P_{QS} starts oscillating (top part), thanks to the positive regulation of the autoinducer (through more frequent formations of the complex $G_{QS} \cdot A$, middle part), whose amount, in this case, does not increase in a linear way (bottom part).

Finally, in Figure 5.14 we show the dynamics within the single cell when all rules (left side), and all rules but the environmental one (right side), are activated. Therefore, in this case, we are also considering the positive regulation of gene G_1 by means of the autoinducer (controlled, in turn, by C, which is assumed to be initially present in 1000 copies inside the cell). On the left side, we show how the amount of P_1 gets notably increased, together with P_{QS} (top part), since there is a high autoinduction of G_{QS} (middle part). The autoinducer shows the same behaviour of the previous figure. On the right side, where the communication of the autoinducer from the environment towards the cell is silenced, we see that the autoinductive regulation of G_{QS} is not triggered (middle part), since few molecules A remain inside the cell (bottom part). Therefore, the dynamics of proteins P_1, P_2, P_3, P_{QS} resembles the normal oscillations of Figure 5.12.

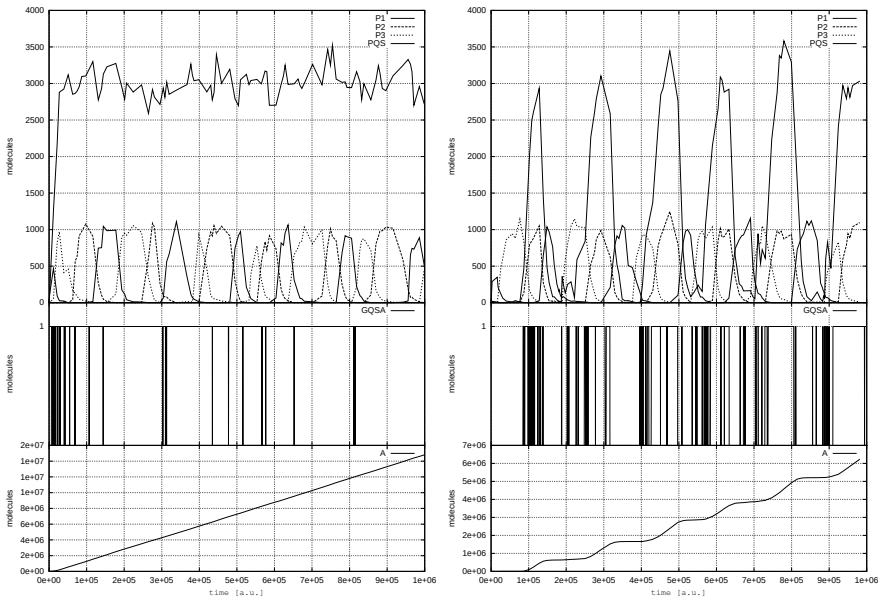


Figure 5.13: Dynamics of quorum sensing protein P_{QS} and autoinducer. Left side: steady state. Right side: controlled oscillations.

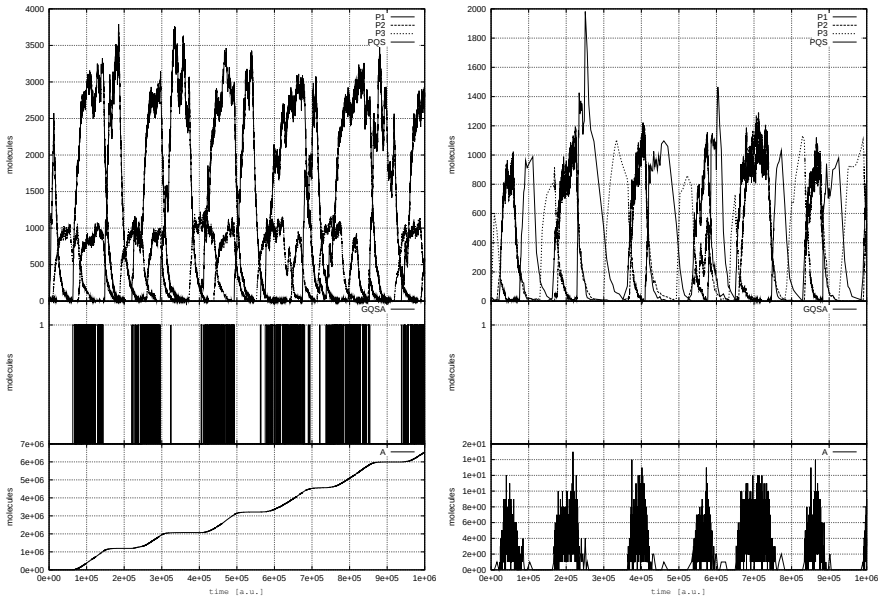


Figure 5.14: Genetic oscillator interlaced with quorum sensing circuit inside the single cell. Left side: active autoinduction. Right side: no autoinduction.

In order to better validate the effective application of communication rules in τ-DPP, and to investigate the role of the intercellular signalling mechanism, we consider a simple example where two cells (plus the environment) are coupled together. In particular, we analyse the case where one cell, say C_1, is unable to produce the autoinducer (that is, rule r_{23} is silenced by setting $c_{23} = 0$), while the other cell (C_2) is fully functional. Note that, if C_1 is placed in an empty environment (where neither cells nor autoinducer molecules are present) under these conditions, then all its internal rules triggered by the autoinducer (namely, $r_{24}, \ldots, r_{34}$) cannot be switched on. Instead, by placing C_1 together with C_2, we show how the correct application of the communication of the autoinducer from C_2 to the environment, and from here to C_1, is able to activate all other rules $r_{24}, \ldots, r_{34}$ in C_1. Therefore, the intercellular (quorum sensing like) mechanism is effectively working in a proper way. The results of this simulation are shown in Figure 5.15 where, on the left side, we present the oscillations occurring along a common time line in cells C_1 (top part, where P_{QS} is not as high as P_1) and C_2 (bottom part, where both P_{QS} and P_1 are elicited). On the right side, instead, we show how frequently the gene G_{QS} gets negatively regulated by P_1 and positively regulated by A (note that the value of $G_{QS} \cdot A$ is multiplied by 1.1 in order to make the graphic more readable).

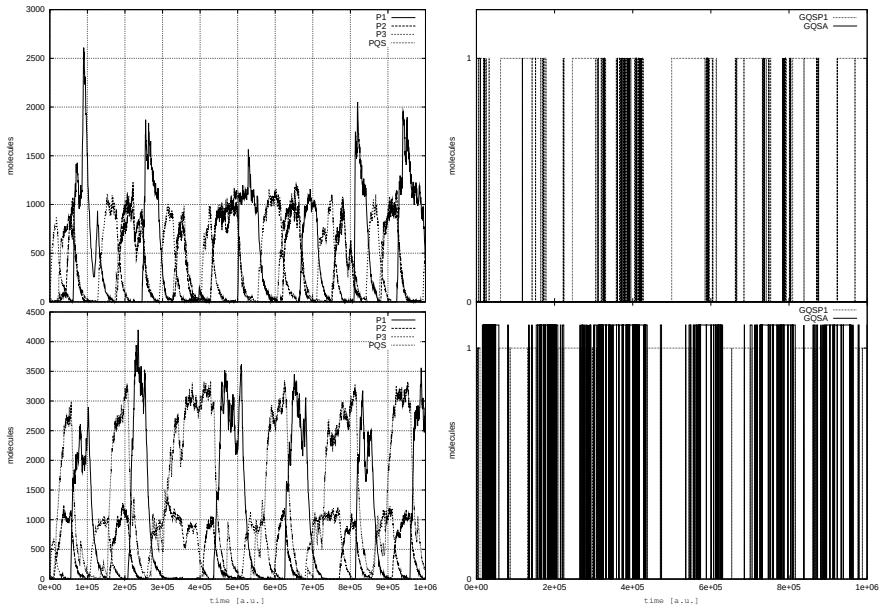


Figure 5.15: Comparison of dynamics between two coupled cells. Left side: oscillations in C_1 (top) and C_2 (bottom). Right side: gene regulation in C_1 (top) and C_2 (bottom).

5.2.3 Discussion and future developments

The investigation of this system allows us to make some considerations on one of the key issues that are related to the simulation of biological systems by means of τ-DPP, that is, the role of stochastic constants associated with communication rules. Usually, in τ-DPP application of a communication rule means that an object (or a set of objects) is *instantaneously* sent to the target volume, without considering the "time" that the objects could need to move from one volume to another one. For instance, a communication rule might represent the passage of a solute through a cellular membrane mediated by a transport protein. This biological process can require a conformational change in the transport protein structure, which necessarily prolong the time the solute would take to virtually cross the membrane in a direct way [146]. Since in τ-DPP we are not considering intermediate "passage structures", a tuning of the stochastic constants of the communication rules is necessary, in order to better represent the behaviour of the objects that move between adjacent volumes.

Furthermore, it would be interesting to check whether a more complex system, consisting of a population of many coupled cells, can show (emergent) synchronization events with respect to the oscillations of the three repressor proteins and the quorum sensing protein. Since the modelling and simulation approach we use here is stochastic, the gain of such a synchronized behaviour in a noisy system is harder than doing this with a deterministic approach (see, e.g., [67]). We will deal with this issue in a future development of this work.

Another extension of this work might consist in considering more complex systems, with populations of different kind of bacteria which oscillate according to a predator–prey dynamics. An example of such kind of systems has been introduced in [9], where the authors presented a synthetic ecosystem composed by two (different) *E. coli* populations, able to communicate by means of a quorum sensing mechanism. In particular, the interactions between the two species regulate each other's gene expression and survival. Indeed, the predator bacteria can kill the prey by inducing the expression of a killer protein in the prey, while preys save predators from committing "suicide" by eliciting the expression of an antidote protein in the predators.

These features might be considered also in a more structured system, i.e. a metapopulation (see Section 3.3), in order to investigate the different dynamics of the bacterial populations with respect to the topological structure of the environment where they interact.

5.3 First steps toward a wet implementation for τ-DPP

The aim of this section is to propose a correspondence between τ-DPP and chemical reacting systems occurring inside micro reactors. Micro reactors [87] are laboratory devices consisting of several reacting volumes (reactors) with a size at the scale of the μl, connected by channels used to transport molecules. Indeed, there is a close relation between the topological description of these two systems: they are both composed by several volumes, and among these volumes it is possible to communicate molecules. Moreover, the approach based on multiset rewriting rules, that characterises τ-DPP, is similar to the chemical reacting process occurring within a micro reactor. Furthermore, both τ-DPP and chemical reacting systems emphasise the intrinsic stochasticity of chemical processes. The "noise", associated to the stochastic behaviour, rules the system dynamics at the micro-scales. At this scale, the small volumes and the high dilutions result in a system where particles interaction should be described in a discrete fashion. Finally, the communication processes described by means of communication rules within τ-DPP, are strictly related to the channels interconnecting the reactors.

In this section, we show how these analogies can be exploited to propose a feasible wet implementation of τ-DPP. To obtain a description of τ-DPP that can be implemented using micro reactors in a straightforward way, we consider the encoding of boolean functions, Fredkin gates and circuits through chemical reactions, following the *chemical computing principles*.

Chemical computing [49] is a technique used to process information by means of real molecules, modified by chemical reactions, or by using electronic devices that are programmed following some principles coming from chemistry. Moreover, in chemical computing, the result of a computation is represented by the emergent global behaviour, obtained from the application of small systems characterised by chemical reactions.

Exploiting the chemical organisation theory [122], we can define a chemical network in order to describe a system as a collection of reactions applied to a given set of molecular species. Moreover, we can identify the set of (so called) organisations, in the set of molecular species, to describe the behaviour of such system. So doing, the behaviour is traced by means of "movement" between organisations.

Furthermore, a different kind of problem encoding will be presented, that is based on the instructions of register machines [134]. This approach is similar to the one related to the chemical computing field. The idea is to use a set of chemical reactions to realize the instructions of the register machines. For instance, in Section 5.3.2, the formalisation and the simulation of a decrement instruction by means of τ-DPP, is presented.

5.3.1 Chemical computing

Here we introduce the basic notions of chemical computing, a novel computational paradigm where the information is processed by means of chemical reactions. Then, we recall the basic definitions of chemical organisation theory, that can be used gain additional knowledge on a chemical computing system, such as its emergent behaviour.

In any generic biochemical system, information is processed by means of chemical reactions occurring between "elementary" components (i.e. molecules); by exploiting this metaphor, chemical reactions can be used to build a novel computational paradigm [49]. This approach is called *chemical computing*, and it is related to the ways of performing computations with both real molecules and electronic devices, which are programmed using principles taken from chemistry.

In general, the analysis of the outcome of chemical reaction processes is hard because of their nonlinearity. The same problems are related to the analysis of biological systems, since the global behaviour can be very different from behaviour of their local components.

In order to work out this problem, the notions of chemical organisation theory can be used to analyse the emergent behaviour of the system, starting from its smaller components; so doing, it is possible to link the evolution governed by the set of all single reactions to the global dynamics of the system.

Chemical organisation theory [122] is used to identify a hierarchy of self maintaining sub-networks, belonging to a chemical reaction network. These sub-networks are called organisations. In particular, a chemical organisation is a set of molecular species that satisfies two properties, that is, it is algebraically closed and stoichiometrically self-maintaining. Here we report an informal definition of these concepts, and refer the reader to [122] for formal definitions and further details.

A *reaction network* is a tuple $\langle \mathcal{M}, \mathcal{R} \rangle$, where $\mathcal{M}$ is a set of molecular species and $\mathcal{R}$ is a set of reactions (also called rules). The rules in $\mathcal{R}$ are given by the relation $\mathcal{R} : P_{\mathcal{M}}(\mathcal{M}) \times P_{\mathcal{M}}(\mathcal{M})$, where $P_{\mathcal{M}}(\mathcal{M})$ denotes the set of all the multisets of elements in $\mathcal{M}$. The general form of a rule is $\alpha_1 m_1 + \alpha_2 m_2 + \cdots + \alpha_k m_k \rightarrow \beta_1 m_1 + \beta_2 m_2 + \cdots + \beta_k m_k$, where $m_1, \ldots, m_k \in \mathcal{M}$ are the molecular species involved in the rule and $\alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_k \in \mathbb{N}$ are the coefficients associated to the molecules.

A set of molecular species $\mathcal{C} \in \mathcal{R}$ is *closed*, if its elements are involved in reactions that produce only molecular species occurring in $\mathcal{C}$. The *self-maintenance* property is satisfied when the molecules in $\mathcal{C}$ consumed by some reaction, can also be produced by some other rule involving molecular species occurring in the set $\mathcal{C}$. Note that, in order to find the organisations of a chemical network, only stoichiometric information (set of rules) is needed.

The set of organisations of a chemical network can be exploited to de-

scribe the dynamics of the system, by means of the movement among different organisations. Namely, the dynamics is traced looking at the system state and at the organisations "represented" by the molecules occurring in the system. Therefore, this analysis consists in the study of processes where molecular species appear or disappear from the system (that is, when their amount become positive or go to zero). Note that, only the algebraic analysis of chemical organisations is sufficient in order to obtain this behaviour, and that the evolution of the system can either take place spontaneously or can be induced by means of external events, such as the addition of input molecules.

If we want to use reaction networks to perform computations, we need to assume that a computational problem can be described as a boolean function, which in turn can be computed as a composition of many simple functions (e.g. the binary NAND). Therefore, we will create a reaction network (called boolean network), based on a set of boolean functions and boolean variables.

Consider a set of M boolean functions $F_1, \ldots, F_M$ and a set of N (with $N \geq M$) boolean variables $b_1, \ldots, b_M, \ldots, b_N$. The variables b_j, $1 \leq j \leq M$, are determined by the boolean functions (they are also called internal variables). The remaining variables (b_j such that $M < j \leq N$) represent the input variables of the boolean network. The values computed by the M boolean functions, are defined as $b_i = F_i(b_{q(i,1)}, \ldots, b_{q(i,n_i)})$, where $i = 1, \ldots, M$ and $b_{q(i,k)}$ is the value of the boolean variable corresponding to the k-th argument of the i-th function. In general, the function F_i has n_i arguments, therefore, there are 2^{n_i} different input combinations.

Given a boolean network of this type, the associated reaction network $\langle \mathcal{M}, \mathcal{R} \rangle$, as presented in [122], is defined as follows. For each boolean variable b_j, two different molecular species, representing the values 0 and 1 of the variable, are added to $\mathcal{M}$. In particular, lowercase letters are used for the molecular species representing the value 0 and uppercase letters for the value 1 of the variables. Therefore, the set $\mathcal{M}$ contains $2N$ molecular species. The set $\mathcal{R}$ of rules is composed by two kinds of reactions: *logical* and *destructive*. Logical reactions are related to the rows of the truth tables of the functions involved in the boolean network; hence, the left-hand side of the rule represents the input values of the boolean function, while the right-hand side is the output value. The destructive reactions are needed to avoid the possibility to have, inside the system, two molecular species representing both states of the same variable at the same time (i.e. two molecules representing the state 0 and 1 of the same boolean variable).

The resulting chemical network $\langle \mathcal{M}, \mathcal{R} \rangle$ implements the boolean network without any specified input. The input variables of the boolean network must be externally initialised because they are not set by the boolean functions. The initialisation is encoded by means of inflow reactions. These reactions are zero-order reactions producing molecules from the empty set.

5.3.2 Definition and simulation of *component reaction networks* using τ-DPP

To lay out our path from a model of computation to a chemical computing device, we define and simulate some test case systems using techniques inspired by the literature on reaction systems [49, 122]. These simple systems must be powerful enough to compute, when assembled in more complex combination, any computable (boolean) function. Hence, we first present the implementations and the simulations of the NAND and XOR logic circuits through sets of chemical reactions, and then more complex systems realised by means of Fredkin gates and circuits.

Finally, we show how we can implement two instructions of register machines by means of chemical reactions: `SUB` and `ADD` (note that, the add instruction will be modelled with in the `SUBADD` module). We recall that *register machines* [134] are universal abstract computing devices, where a finite set of uniquely labelled instructions is given, and which keep updated a finite set of registers (holding integer numbers) at any time by performing a sequence of instructions, chosen according to their labels. Every instruction can be of one of the following type, here informally described:

- `ADD`: a specified register is increased by 1, and the label of the next instruction is nondeterministically chosen between two labels specified in the last instruction applied;
- `SUB`: a specified register is checked, and if it is non-empty, then it is decreased by 1, otherwise it will not be changed; the next label will be differently chosen in the two cases;
- `HALT`: the machine stops.

A boolean logic gate can be described through an appropriate set of molecular species manipulated by some chemical reactions. The molecular species are used to codify the 0/1 values of the logical variables; therefore, at least two species are needed for each gate line, plus some possible auxiliary species that are necessary to control the gate's functioning. The chemical reactions, by consuming and producing the chemicals in a specified way, are used to implement the boolean function that transforms each input into the correct output of the gate. The network of molecular species and chemical reactions implementing a single logic gate can then be extended to describe logic circuits. This extension necessarily requires to manipulate in a coherent way the output and input species of all interconnected gates, in such a way that the functioning of each single gate is properly simulated and, at the same time, the correct flow of information through the whole circuit is guaranteed.

To simulate the functioning of the logic gate by means of chemical reactions, three types of internal reactions are used. The so called *input re-*

actions are needed to produce the input chemicals inside the volume, thus simulating the input bits that are given to the gate. The *logical reactions* are related to the rows of the truth table of the gate: they describe how the output chemicals are produced whenever the corresponding input chemicals appear inside the reaction volume. The *degradation reactions* are needed to avoid the simultaneous presence of two conflicting species inside the volume, representing both the states 0 and 1 on the same gate line (as discussed hereby).

When two or more logic gates are connected in a logic circuits, some *communication reactions* are needed in order to simulate the lines that actually connect the gates. Communication reactions send output molecules from one volume to the next one to simulate the information propagated through the circuit.

We believe that τ-DPP represents a feasible framework for this task, as it can be used to describe the chemical network underlying single logic gates – as well as the more complex structure of a circuit – and to simulate their functioning by following the temporal evolution of the input and output molecular species.

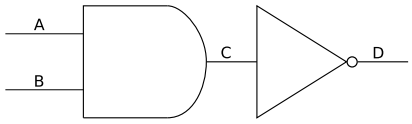
5.3.2.1 The NAND and XOR logic circuits

The NAND logic circuit has been implemented with the sequential composition of an AND and a NOT gate as shown in Figure 5.16 (left). Following the chemical computing guidelines described in Section 5.3.1, we define the logic circuit with the rules listed in Figure 5.16 (right). Rules $r_1, \ldots, r_4$ compute the AND function, rules $r_5, \ldots, r_6$ compute the NOT function and rules $r_7, \ldots, r_{10}$ "clean" the system when both values of a variable are present at the same time, as described in Section 5.3.1. Finally, rules $r_{11}, \ldots, r_{14}$ represent the inputs of the gate because they produce the molecules a, A, b and B, representing the inputs $A = 0$, $A = 1$, $B = 0$ and $B = 1$ of the NAND logic circuit, respectively. For instance, when the constants of the rules r_{11} and r_{13} are set to 1, the input given to the NAND gate is 0 for both the input lines because molecules a and b are produced. The rationale behind this, is that the different inputs for the system are obtained producing the molecular species used to represent that particular values. The values of the constants reported in the table have been used to perform the simulation of the NAND behaviour by means of τ-DPP.

Starting from the set of rules presented above for the NAND logic circuit, it is possible to define the τ-DPP which encodes the logic circuit. Formally, the τ-DPP Υ_{NAND} is defined as

$$\Upsilon_{NAND} = (V_0, \mu, \mathcal{S}, M_0, R_0, C_0),$$

where:



Reaction	Constant
$r_1 : a + b \to c$	$c_1 = 1 \cdot 10^{-3}$
$r_2 : a + B \to c$	$c_2 = 1 \cdot 10^{-3}$
$r_3 : A + b \to c$	$c_3 = 1 \cdot 10^{-3}$
$r_4 : A + B \to C$	$c_4 = 1 \cdot 10^{-3}$
$r_5 : c \to D$	$c_5 = 1 \cdot 10^{-2}$
$r_6 : C \to d$	$c_6 = 1 \cdot 10^{-2}$
$r_7 : a + A \to \lambda$	$c_7 = 1 \cdot 10^{-1}$
$r_8 : b + B \to \lambda$	$c_8 = 1 \cdot 10^{-1}$
$r_9 : c + C \to \lambda$	$c_9 = 1 \cdot 10^{-1}$
$r_{10} : d + D \to \lambda$	$c_{10} = 1 \cdot 10^{-1}$
$r_{11} : \lambda \to a$	$c_{11} \in \{1, 0\}$
$r_{12} : \lambda \to A$	$c_{12} \in \{1, 0\}$
$r_{13} : \lambda \to b$	$c_{13} \in \{1, 0\}$
$r_{14} : \lambda \to B$	$c_{14} \in \{1, 0\}$

Figure 5.16: The NAND logic circuit (left) and the set of reactions used to implement it (right).

- V_0 is the unique volume of the NAND logic circuit;
- μ is the membrane structure $[_0 \ ]_0$;
- $\mathcal{S} = \{a, A, b, B, c, C, d, D\}$ is the set of molecular species;
- $M_0 = \emptyset$ is the initial multiset occurring inside the volume V_0;
- $R_0 = \{r_1, \ldots, r_{14}\}$ is the set of rules defined in volume V_0 and reported in Figure 5.16. Due to the membrane structure μ, all the rules here involved are internal.
- $C_0 = \{c_1, \ldots, c_{14}\}$ is the set of stochastic constants associated to the rules defined in R_0, and reported in Figure 5.16.

In Figure 5.17, the result of the simulation of the NAND gate is reported. In the initial configuration of the system, the multiset is empty, that is, the amounts of all molecular species are set to zero. At time $t = 0$, we simulate that the system receives the molecules a, B as input, which corresponds to setting the first input line to zero and the second line to one. This is formulated as a τ-DPP configuration where the constants of rules r_{11} and r_{14} are equal to 1, while the constants of rules r_{12} and r_{13} are equal to 0. The output obtained with this configuration is 1: the system starts producing the molecules D corresponding to the expected output value. At time $t = 400$, the input values of the system are changed from a, B to A, B, setting c_{11} and c_{14} to 0 and c_{12} and c_{13} to 1. Now the system starts producing d molecules, but the output of the system actually changes only when all the D molecules have been degraded (by means of rule r_{10}) and the molecules d are then accumulated inside the membrane.

The XOR logic circuit (left side of Figure 5.18) has been implemented using the set of rules listed in Figure 5.18 (right). The rules $r_1, \ldots, r_4$

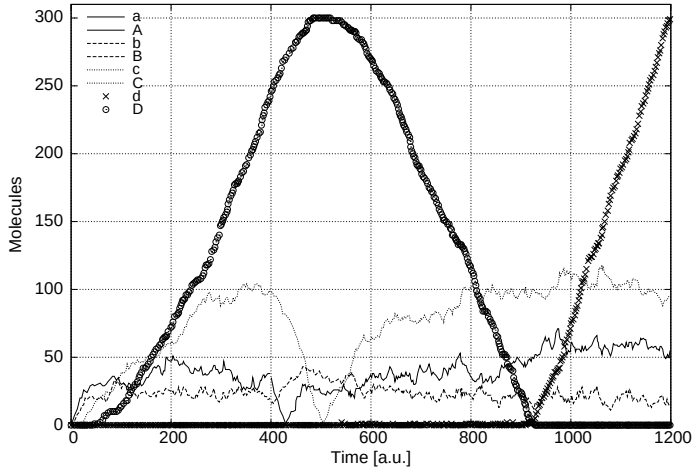
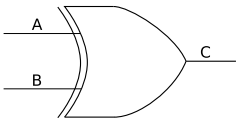


Figure 5.17: Plot of the dynamics of the NAND unit with two inputs given at time $t = 0$ and $t = 400$.



Reaction	**Constant**
$r_1 : a + b \to c$	$c_1 = 1 \cdot 10^{-3}$
$r_2 : a + B \to C$	$c_2 = 1 \cdot 10^{-3}$
$r_3 : A + b \to C$	$c_3 = 1 \cdot 10^{-3}$
$r_4 : A + B \to c$	$c_4 = 1 \cdot 10^{-3}$
$r_5 : a + A \to \lambda$	$c_5 = 1 \cdot 10^{-1}$
$r_6 : b + B \to \lambda$	$c_6 = 1 \cdot 10^{-1}$
$r_7 : c + C \to \lambda$	$c_7 = 1 \cdot 10^{-1}$
$r_8 : \lambda \to a$	$c_8 \in \{1, 0\}$
$r_9 : \lambda \to A$	$c_9 \in \{1, 0\}$
$r_{10} : \lambda \to b$	$c_{10} \in \{1, 0\}$
$r_{11} : \lambda \to B$	$c_{11} \in \{1, 0\}$

Figure 5.18: The XOR logic circuit (left), and set of reactions used to implement it (right).

compute the XOR function and $r_5, \ldots, r_7$ "clean" the system when both values of a variable are present at the same time, as described in Section 5.3.1. Finally, the rules $r_8, \ldots, r_{11}$ represent the inputs of the gate. For instance, when the constants of the rules r_8 and r_{10} are set to 1, the input given to the XOR gate is 0 for both the input lines. The values of the constant reported in the table have been used to perform the simulation by means of τ-DPP.

Formally, the τ-DPP Υ_{XOR}, corresponding to the XOR logic circuit, is defined as

$$\Upsilon_{XOR} = (V_0, \mu, \mathcal{S}, M_0, R_0, C_0),$$

where:

- V_0 is the unique volume of the XOR logic circuit;
- μ is the membrane structure $[_0 \ ]_0$;

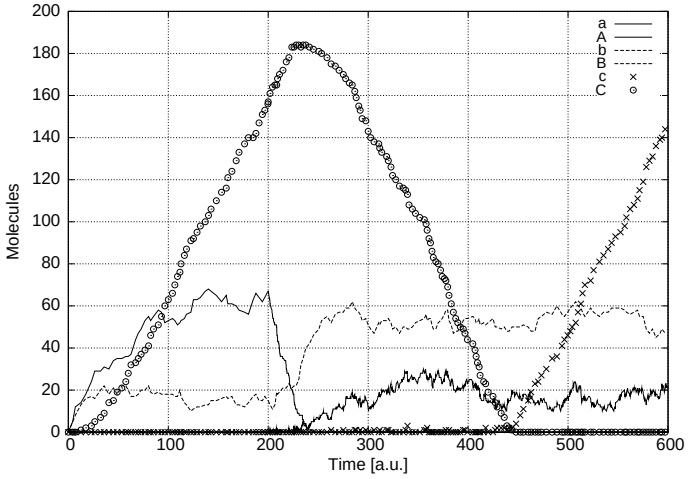


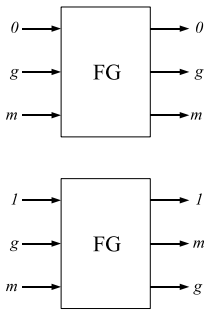
Figure 5.19: Plot of the dynamics of the XOR unit with two inputs given at time $t = 0$ and $t = 200$.

- $\mathcal{S} = \{a, A, b, B, c, C, d, D\}$ is the set of molecular species;
- $M_0 = \emptyset$ is the initial multiset occurring inside the volume V_0;
- $R_0 = \{r_1, \ldots, r_{11}\}$ is the set of rules defined in volumes V_0 and reported in Figure 5.18. Due to the membrane structure μ, all the rules here involved are internal.
- $C_0 = \{c_1, \ldots, c_{11}\}$ is the set of stochastic constants associated to the rules defined in V_0 and reported in Figure 5.18.

In Figure 5.19, the result of the simulation of the XOR gate is reported. In the initial configuration of the system, the multiset is empty, that is, the amounts of all molecular species are set to zero. At time $t = 0$, we simulated that the system receives the molecules a, B as input, which corresponds to setting the first input line to zero and the second one to one. This is formulated as a τ-DPP configuration where the constants of rules r_8 and r_{11} are equal to 1, while the constants of rules r_9 and r_{10} are equal to 0. The output obtained with this configuration is 1: the system producing the molecules C corresponding to the expected output value. At time $t = 200$ the input values of the system are changed from a, B to A, B, setting c_8 to 0 and c_9 to 1. Now the system starts producing c molecules, but the output of the system actually changes only when all the C molecules have been degraded (by means of rule r_7) and the molecules c are then accumulated inside the membrane.

5.3.2.2 Fredkin gates and circuits

The *Fredkin gate* is a three–input/three–output boolean gate, whose input/output map FG : $\{0,1\}^3 \to \{0,1\}^3$ is logically reversible (that is, its inputs can always be deduced from its outputs) and preserves the number of 1's given in input. The map FG associates any input triple $(\alpha_i, \beta_i, \gamma_i)$ with its corresponding output triple $(\alpha_o, \beta_o, \gamma_o)$ according to the formula $\alpha_o = \alpha_i$, $\beta_o = (\neg\alpha_i \wedge \beta_i) \vee (\alpha_i \wedge \gamma_i)$, $\gamma_o = (\alpha_i \wedge \beta_i) \vee (\neg\alpha_i \wedge \gamma_i)$ (see the truth table in Figure 5.20). It is worth noting that the Fredkin gate behaves as a conditional switch, since α_i can be considered as a control line whose value determines whether the input values β_i and γ_i have to be exchanged or not: $\mathrm{FG}(1, \beta_i, \gamma_i) = (1, \gamma_i, \beta_i)$ and $\mathrm{FG}(0, \beta_i, \gamma_i) = (0, \beta_i, \gamma_i)$ for every $\beta_i, \gamma_i \in \{0,1\}$.



$\alpha_i\ \beta_i\ \gamma_i$	$\mapsto$	$\alpha_o\ \beta_o\ \gamma_o$
0 0 0		0 0 0
0 0 1		0 0 1
0 1 0		0 1 0
0 1 1		0 1 1
1 0 0		1 0 0
1 0 1		1 1 0
1 1 0		1 0 1
1 1 1		1 1 1

Figure 5.20: The Fredkin gate: its behaviour as a conditional switch (left) and its truth table (right).

The Fredkin gate is functionally complete for boolean logic: by fixing $\gamma_i = 0$ we obtain $\gamma_o = \alpha_i \wedge \beta_i$, whereas by fixing $\beta_i = 1$ and $\gamma_i = 0$ we obtain $\beta_o = \neg\alpha_i$. By inspecting the truth table, we can see that the Fredkin gate is also logically reversible, since the map FG is a bijection on $\{0,1\}^3$. Moreover, it is conservative: for every input/output pair the number of 1's in the input triple is the same as the number of 1's in the output triple. In other words, the output triple is obtained by applying an appropriate (input–dependent) permutation to the input triple.

The Fredkin gate is at the basis of the model of conservative logic introduced in [64], which describes computations by considering some notable properties of microdynamical laws of physics, such as reversibility and the conservation of the internal energy of the physical system by which computations are performed. Within that model, computations are performed by reversible *Fredkin circuits*, which are obtained by putting together Fredkin gates. The evaluation of a Fredkin circuit in topological order (i.e. layer by layer) defines the boolean function computed by the circuit, which is given by the composition of the functions computed by each layer of Fredkin gates.

The conservativeness of the circuit (preservation of the number of 1's) is equivalent to the requirement that the output n-tuple is obtained by applying an appropriate (input–dependent) permutation to the corresponding input n-tuple.

Simulations results In this section, we first show how we can simulate a Fredkin gate by means of chemical reactions; then, we present an example of a Fredkin circuit, consisting of three gates arranged into two layers, and describe how it can be implemented by means of a tissue-like membrane structure.

Concerning the Fredkin gate, we define two different types of molecular species for each of its lines; as previously stated, we will denote by means of lowercase letters the chemicals representing the value 0, while uppercase letters will be used for the chemicals representing the value 1. In particular, in the following we use: the symbols a, A for both the input α_i and output α_o on the first line of the gate; b, B for the input β_i and d, D for the output β_o on the second line; c, C for the input γ_i and e, E for the output γ_o on the third line. The rationale behind this choice is that, for the first line – where the output bit is always equal to the input bit – two different chemical species suffice, while for the second and the third lines of the gate – where the logic switch is implemented – we will need to distinguish among the 0 and 1 bits given as input or generated as output. All these chemicals will be manipulated inside a single volume of a τ-DPP.

To simulate the functioning of the logic gate by means of chemical reactions, we use input, logical and degradation reactions (no communication rule is needed here).

According to the definition of τ-DPP [30], for each chemical reaction there exists a corresponding stochastic constant that is used to evaluate the probability of applying that reaction, at each time step, according to the current configuration of the system. In the case of the Fredkin gate, we have the additional peculiarity that the constants associated to the input reactions can be modified – actually, turned on and off – at chosen time instants during the system evolution (while the constants of logical and degradation reactions have a value that is fixed during the system evolution, as usual in τ-DPP). So doing, we can simulate the possibility that the input lines $\alpha_i, \beta_i, \gamma_i$ of the Fredkin gate can receive a different bit at any chosen time instant. The process can then be repeated at successive time instants, thus allowing the gate to receive any sequence of input bits. Nevertheless, constraints for the activation of input reactions are to be given, in order to avoid the presence of conflicting input species inside the volume, which would correspond to have the two opposite bits on the same input line simultaneously. The constraints are the following: if an input reaction $r_{i=0}$ for bit 0 is turned on for one of the three lines of the gate (that is, if species x

Table 5.5: Chemical reactions for the Fredkin gate.

Reaction	Constant
$r_1 : a + b \to a + d$	$c_1 = 1 \cdot 10^{-3}$
$r_2 : a + c \to a + e$	$c_2 = 1 \cdot 10^{-3}$
$r_3 : a + B \to a + D$	$c_3 = 1 \cdot 10^{-3}$
$r_4 : a + C \to a + E$	$c_4 = 1 \cdot 10^{-3}$
$r_5 : A + b \to A'$	$c_5 = 1 \cdot 10^{-3}$
$r_6 : A' + c \to A + d + e$	$c_6 = 1 \cdot 10^{-1}$
$r_7 : A' + C \to A + D + e$	$c_7 = 1 \cdot 10^{-1}$
$r_8 : A + B \to A''$	$c_8 = 1 \cdot 10^{-3}$
$r_9 : A'' + c \to A + d + E$	$c_9 = 1 \cdot 10^{-1}$
$r_{10} : A'' + C \to A + D + E$	$c_{10} = 1 \cdot 10^{-1}$

Reaction	Constant
$r_{11} : a + A \to \lambda$	$c_{11} = 1 \cdot 10^{-1}$
$r_{12} : b + B \to \lambda$	$c_{12} = 1 \cdot 10^{-1}$
$r_{13} : c + C \to \lambda$	$c_{13} = 1 \cdot 10^{-1}$
$r_{14} : d + D \to \lambda$	$c_{14} = 1 \cdot 10^{-1}$
$r_{15} : e + E \to \lambda$	$c_{15} = 1 \cdot 10^{-1}$
$r_{16} : \lambda \to a$	$c_{16} \in \{0, 1\}$
$r_{17} : \lambda \to A$	$c_{17} \in \{0, 1\}$
$r_{18} : \lambda \to b$	$c_{18} \in \{0, 1\}$
$r_{19} : \lambda \to B$	$c_{19} \in \{0, 1\}$
$r_{20} : \lambda \to c$	$c_{20} \in \{0, 1\}$
$r_{21} : \lambda \to C$	$c_{21} \in \{0, 1\}$

is produced inside the volume, where $x = a, b$ or c), then the input reaction $r_{i=1}$ for bit 1 of the same line has to be turned off (that is, species X cannot be produced inside the volume, where $X = A, B$ or C, respectively). In terms of constants, this means that, in general, if $c_{r_{i=0}} = k$, for some $k > 0$, then $c_{r_{i=1}} = 0$; on the contrary, if $c_{r_{i=0}} = 0$ then $c_{r_{i=1}} = k$ (in the following, we always assume $k = 1$).

We stress the fact that straight after the time instant when the inputs are changed, the system is usually in a configuration where some copies of both species x, X (for $x = a, b, c, d$ or e and, correspondingly, $X = A, B, C, D$ or E) occur inside the reaction volume. To solve this contradiction, degradation reactions are expressly defined inside each volume in order to remove the conflicting species.

Formally, we present the τ-DPP specification for simulating a single Fredkin gate. We consider the construct $\Upsilon_{FG} = (V, \mathcal{S}, M, R, C)$ where:

- V is the single reaction volume corresponding to the Fredkin gate;
- $\mathcal{S} = \{a, A, b, B, c, C, d, D, e, E, A', A''\}$ is the set of molecular species;
- $M = \emptyset$ is the initial multiset inside V;
- $R = \{r_1, \ldots, r_{21}\}$ is the set of chemical reactions defined inside V, and reported in Table 5.5;
- $C = \{c_1, \ldots, c_{21}\}$ is the set of stochastic constants associated to the reactions in R, and reported in Table 5.5.

Reactions $r_1, \ldots, r_{10}$ are the logical chemical reactions which specify how the output chemicals are produced, and which simulate that a given input triple occurs on the lines $\alpha_i, \beta_i, \gamma_i$ of the Fredkin gate. In particular, reactions $r_1, \ldots, r_4$ produce the output chemicals d, D and e, E when the input chemical on α_i is equal to $a = 0$, thus simulating bits 0 and 1 maintained on β_o and γ_o, respectively (that is, when no switch has to take place). On the contrary, when the input on α_i is equal to 1 (this case is simulated through

the chemical A) a switch must be performed on the output values of the second and the third line. In this case, we need to exploit two auxiliary chemicals, denoted by A' and A'', in order to correctly generate the output chemicals for β_o and γ_o. In particular, A' is used to simulate the presence of bit 0 on β_i (thus allowing to produce the output chemicals corresponding to the input triple $(1, 0, \gamma_i)$, for any $\gamma_i \in \{0, 1\}$), while A'' is used to simulate the presence of bit 1 on β_i (thus allowing to produce the output chemicals corresponding to the input triple $(1, 1, \gamma_i)$, for any $\gamma_i \in \{0, 1\}$). As soon as these auxiliary symbols are produced inside the reaction volume, the reactions r_6 and r_9 (r_7 and r_{10}) generate the correct output chemicals when the input of γ_i is equal to 0 (1, respectively).

Reactions $r_{11}, \ldots, r_{15}$ are the degradation reactions that can only be applied whenever at least one copy of each conflicting (couple of) chemicals x and X occur inside the reaction volume. In other words, these reactions are applied when the input values of the Fredkin gate are changed.

Reactions $r_{16}, \ldots, r_{21}$ are the input reactions that produce the input chemicals inside the volume, by mimicking the addition of a chemical solution into the reaction volume. At any time step, exactly three of these reactions are turned on, such that if $c_i \neq 0$ then $c_{i+1} = 0$, and if $c_i = 0$ then $c_{i+1} \neq 0$, for any $i \in \{16, 18, 20\}$.

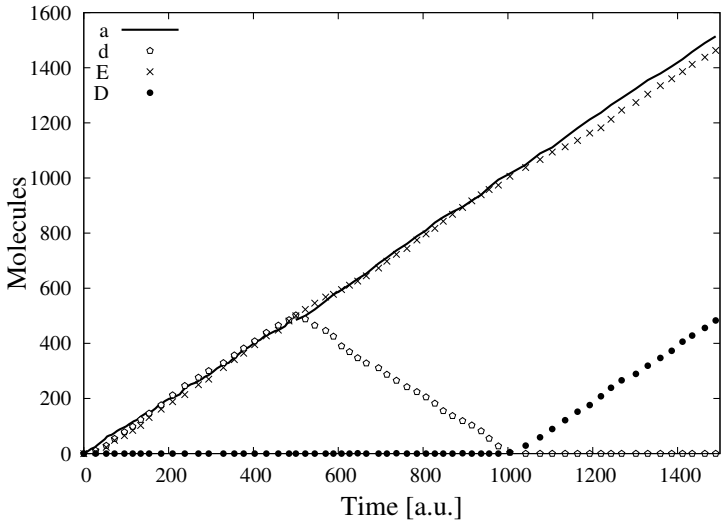


Figure 5.21: Simulations of the Fredkin gate: output results for the first input $(\alpha_i, \beta_i, \gamma_i) = (0, 0, 1)$ at $t = 0$, and the second input $(\alpha_i, \beta_i, \gamma_i) = (0, 1, 1)$ at $t = 500$.

In Figures 5.21 and 5.22 we present two independent simulations of the Fredkin gate; in each one, two different input triples have been given at distinct time instants. Namely, in Figure 5.21 we show the simulation

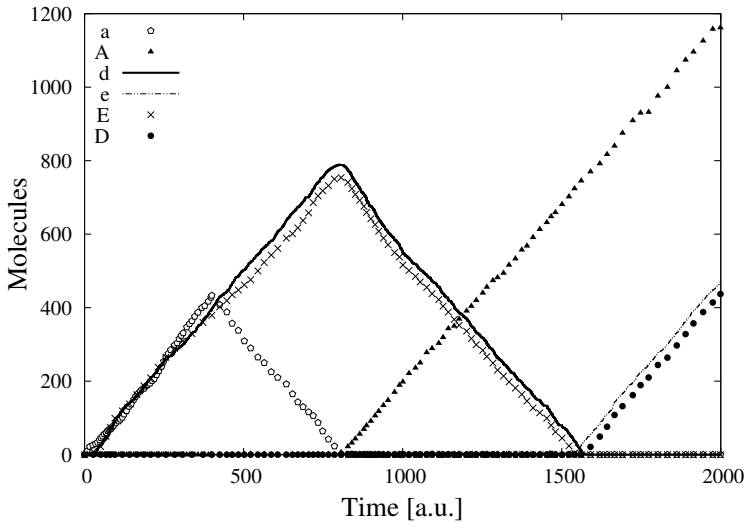


Figure 5.22: Simulations of the Fredkin gate: output results for the first input $(\alpha_i, \beta_i, \gamma_i) = (0, 0, 1)$ at $t = 0$, and the second input $(\alpha_i, \beta_i, \gamma_i) = (1, 0, 1)$ at $t = 400$.

results of a Fredkin gate where the first input corresponds to the triple $(\alpha_i, \beta_i, \gamma_i) = (0, 0, 1)$ given at the initial time $t = 0$, while the second input is given at the time instant $t = 500$ and corresponds to the triple $(\alpha_i, \beta_i, \gamma_i) = (0, 1, 1)$. According to the truth table of the Fredkin gate (Figure 5.20), when the first input is received, the corresponding output triple has to be $(\alpha_o, \beta_o, \gamma_o) = (0, 0, 1)$; translated into molecular species, this output corresponds to the chemicals d and E produced inside the volume. This behaviour is apparent from the linear growth of these two species in the time interval $t \in [0, 500]$. At $t = 500$, when the input value on the second line is changed by turning on reaction r_{19} (and turning off the conflicting reaction r_{18}), the molecular species D starts to be produced by means of rule r_3. Therefore, the system generates a configuration where some copies of both species d and D (values 0 and 1 on β_o) simultaneously occur inside the volume. To solve this contradiction, the output species d and D for β_o are degraded by applying reaction r_{14}, along with the corresponding input species b and B for β_i, which are degraded by applying reaction r_{12}. Once all copies of species d are erased from the volume (around $t = 1000$), the number of copies of species D increases, thus generating the correct second output for the second line. We outline that, though not explicitly evident from the figure, in the time interval $t \in [500, 1000]$ the number of copies of D are not exactly equal to zero but are characterized by small positive fluctuations (of the order of 2-3 molecules per time step). This behaviour is due to the choice of the stochastic constants, as given in Table 5.5, whereby

the degradation reaction r_{14} (for species d and D) has a higher probability of being applied with respect to the input reaction r_3 (which produces the output species D). Therefore, as soon as a molecule of species D appears inside the volume, it is almost immediately consumed by reaction r_{14}. So doing, we allow a more rapid decrease of the first output species d and the consecutive increase of the second output species D.

In Figure 5.22 we show the simulation results of the Fredkin gate with first input triple equal to $(\alpha_i, \beta_i, \gamma_i) = (0, 0, 1)$, given at the time instant $t = 0$, and with second input triple equal to $(\alpha_i, \beta_i, \gamma_i) = (1, 0, 1)$, given at time instant $t = 400$. Since in this case we are changing the input bit on the first line of the gate, the switch between the second and the third line will be implemented; therefore, all the three output molecular species A, D and e generated after the second input will be different from the output species a, d and E generated after the initial input. In the picture we represent the temporal evolution of all these six chemicals. During the time interval $t \in [0, 400]$, we can observe the linear growth of species a, d, and E. At time $t = 400$, when the bit on α_i is changed from 0 to 1 (reaction r_{17} is turned on, and r_{16} is turned off), we see that the amount of a starts to decrease due to the application of reaction r_{11}, until it is completely exhausted (around $t = 800$), and the amount of chemical A can now increase. At this time, also the production of output species d and E stops since, due to the presence of the species A inside the volume, reactions r_1 and r_4 cannot be applied anymore, and the copies of d and E still present in the volume start to get erased by applying reactions r_{14} and r_{15}. When all the copies of species d and E are degraded (around $t = 1600$), the actual output species D and e, corresponding to the second output, start to grow. Therefore, we can correctly simulate the functioning of the Fredkin gate.

The chemical behaviour of the Fredkin gate highlights one of the biggest drawbacks of simulating logic gates with chemical reactions: whenever the input is changed, a time delay elapses before the corresponding output can be effectively generated. Anyway, this delay can be reduced and modulated by appropriately adjusting the stochastic constants of the chemical reactions and, in particular, of the degradation reactions with respect to the input reactions. Stated in other words, when a new input is provided to the gate, in molecular terms there exists an average degradation time before the reaction volume can be cleared out of the *old* output chemicals, in order for the *new* output chemicals to increase and hence produce the expected outcome. In principle, we can give an average measure of this delay according to the stochastic constants chosen for all reactions, therefore deriving and controlling the average time interval that has to elapse before the expected output can be read.

We now present some simulations of the circuit depicted on the left side of Figure 5.23, consisting of three Fredkin gates, structured into two layers (where we denote by $x_1, \ldots, x_7$ and $y_1, \ldots, y_7$ the input and output lines of

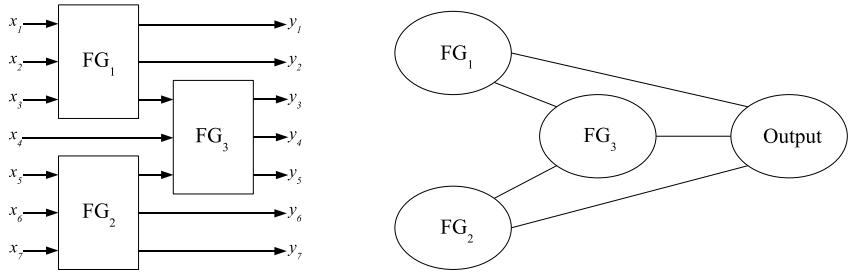


Figure 5.23: A Fredkin circuit with three gates and two layers (left) and its tissue-like representation (right).

the circuit, respectively). On the right side of this figure, we also represent the tissue-like structure of τ-DPP that is used here to simulate the entire circuit: we connect three reaction volumes (called FG_1, FG_2, FG_3) in such a way that the output chemicals of lines γ_o^1 and α_o^2 of gates 1 and 2 in the first circuit layer correspond to the input chemicals of lines α_i^3 and γ_i^3 of gate 3, respectively (here, the superscripts on line names clearly denote the number of the gate). Moreover, we consider an additional reaction volume whose function is to collect all output species of each gate. Each reaction volume FG_i corresponds to the definition of τ-DPP Υ_{FG} describing a single Fredkin gate, with the following differences:

- in Υ_{FG_1}, communication rules are used to send the output chemicals a, A and d, D of lines y_1, y_2 to the output volume, and the output chemicals e, E of the third line (that represents the first input line for FG_3) to volume FG_3;
- in Υ_{FG_2}, communication rules are used to send the output chemicals d, D and e, E of lines y_6, y_7 to the output volume, and the output chemicals a, A of the fifth line (that represents the third input line for FG_3) to volume FG_3;
- in Υ_{FG_3}, communication rules are used to send all output chemicals to the output volume, and only the input reactions of the second line can be turned on (since the first and third lines receive the input from FG_1 and FG_2, respectively).

The output volume, instead, contains only the degradation reactions defined in the standard definition of Υ_{FG}, which are needed to erase the conflicting output chemicals whenever the input to the circuit is changed. So doing, we guarantee that the correct result of the entire circuit can be read inside a unique volume.

In Figure 5.24 we show a simulation of this Fredkin circuit, by splitting all output chemicals in three distinct panels for a better comprehension of their temporal evolution. During the simulation, the circuit receives a

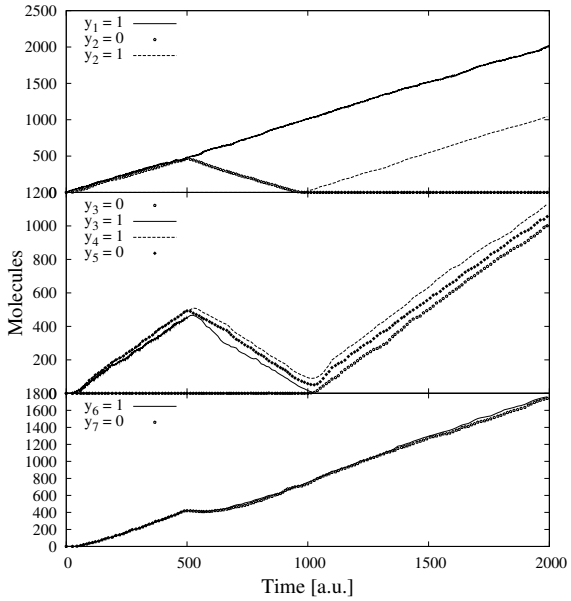


Figure 5.24: Simulation of the Fredkin circuit given in Figure 5.23. Output results with first input $(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (1, 1, 0, 0, 1, 0, 1)$ at $t = 0$, and second input $(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (1, 0, 1, 1, 0, 1, 0)$ at $t = 500$.

first input $(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (1, 1, 0, 0, 1, 0, 1)$ at time $t = 0$, and then a second input $(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (1, 0, 1, 1, 0, 1, 0)$ at time $t = 500$. These two inputs generate the two outputs $(y_1, y_2, y_3, y_4, y_5, y_6, y_7) = (1, 0, 1, 1, 0, 1, 0)$ and $(y_1, y_2, y_3, y_4, y_5, y_6, y_7) = (1, 1, 0, 1, 0, 1, 0)$, respectively. Note that in the latter case only the bits on lines y_2 and y_3 are changed. During the first time interval $t \in [0, 500]$, the output chemicals corresponding to the first input are first generated inside the three reaction volumes FG_1, FG_2 and FG_3 and then, by using the communication rules defined in $\Upsilon_{FG_1}, \Upsilon_{FG_2}, \Upsilon_{FG_3}$, all these output chemicals are collected in the output volume. At time $t = 500$ all lines of the circuit, except the first one, receive a different bit: this change results in the variation of two output bits on the lines y_2 and y_3, and the corresponding production of two new output chemicals in reaction volumes FG_1 and FG_3. The dynamics of these two output chemicals can be seen in the top and central panels of Figure 5.24, respectively, where it is apparent how the chemicals corresponding to the output bits $y_2 = 1$ and $y_3 = 0$ start to increase as soon as their conflicting chemicals (corresponding to the old output bits $y_2 = 0$ and $y_3 = 1$) are completely degraded. We also highlight the behaviour of the output chemicals associated to lines $y_4, \ldots, y_7$ in the central and bottom panels of Figure 5.24: even if the values of these bits are not affected by the second input, their temporal evolution does not follow the expected linear growth. This is due

to the fact that, when changing the value of x_5 from the first to the second input, the degradation delay required to consume all conflicting species is now impacting on the dynamics of the output lines y_6 and y_7 in a weak way (they just show a smooth and fast decrease before they can start to grow again, as shown in the bottom panel), and of the output lines y_4 and y_5 in a sharper way (they follow a dynamics similar to that of y_3, as shown in the central panel).

It is thus clear that, similarly to the single Fredkin gate, when simulating a change of the input value in a circuit, we have to wait for species degradation before the effective new output is generated. This time, however, the topology of the circuit (number of layers and gates, and their interconnections) brings about an additional "noise" on the temporal evolution of chemicals. This is due to an imbalance of the quantity of the chemical species corresponding to the lines that are not connected to other gates in a successive layer (e.g. the first and second lines of FG_1, and the second and third lines of FG_2 in Figure 5.23), with respect to the quantity of the species corresponding to the lines that become input for some other gate in a successive layer (e.g. the third line of FG_1 and the first line of FG_2 in Figure 5.23). Such imbalance could be avoided by providing a different form of the circuit. In fact, in [105] it is shown how to transform any Fredkin circuit, consisting of a given number of gates arranged in a fixed configuration of layers, into an equivalent (normalized) Fredkin circuit, where each layer contains the same number of gates (to this aim, auxiliary gates and lines are used). By considering this kind of normalized circuits, it is possible to simulate their functioning through chemical reaction systems without having to deal with the aforementioned problems of degradation delays. However, in this case we would have to handle more complex systems, consisting of more volumes and many more chemical species and reactions.

5.3.2.3 The SUB instruction

We now describe and simulate a τ-DPP composed by 2 volumes reproducing the behaviour of a `SUB` instruction of a register machine. This type of instruction is important because it hides a conditional behaviour, checking whether a register is zero or not, and can be implemented by choosing a different label for the next instruction according to the register value. The availability of conditional instructions is a key issue in computing devices.

We implement a system where the quantity stored inside the register is represented by the amount of molecules u occurring in volume V_1. The label for the next instruction is related to the production of molecules p or z in volume V_0. Finally, to start the system, molecules s are produced inside V_0. s' and z' are additional molecular species used to implement the instruction.

In order to correctly execute the `SUB` instruction, when molecules s are sent inside the volume V_1, the system first checks if the register value is

zero, that is, if any u molecule occurs inside V_1; then, the label for the next instruction is produced.

Formally, a τ-DPP Υ_{SUB} is defined as

$$\Upsilon_{SUB} = (V_0, V_1, \mu, \mathcal{S}_\prime, \mathcal{S}_\infty, M_0, M_1, R_0, R_1, C_0, C_1),$$

where:

- V_0, V_1 are the volumes of the SUB unit;
- μ is the nested membrane structure $[_0\ [_1\ \ ]_1\ ]_0$;
- $\mathcal{S}_0 = \{p, s, s', z\}$ and $\mathcal{S}_1 = \{p, s, u, z, z'\}$ are the sets of molecular species of volumes V_0 and V_1, respectively;
- $M_0 = \{s'^{40}\},\ M_1 = \{u^{20}, z^5\}$, are the multisets occurring inside the membranes V_0 and V_1 at time $t = 0$;
- $R_0 = \{r_{0_1}, \ldots, r_{0_5}\},\ R_1 = \{r_{1_1}, \ldots, r_{1_3}\}$ are the sets of rules defined in volumes V_0, V_1, respectively, and reported in Table 5.6;
- $C_0 = \{c_{0_1}, \ldots, c_{0_5}\},\ C_1 = \{c_{1_1}, \ldots, c_{1_3}\}$ is the sets of stochastic constants associated to the rules defined in V_0 and V_1, respectively, and reported in Table 5.6.

Table 5.6: Reactions for the SUB unit (R_0 on the left and R_1 on the right).

Reaction	Constant
$r_{0_1} : 2p \to (p, here)$	$c_{0_1} = 1$
$r_{0_2} : z + p \to (z, here)$	$c_{0_2} = 1$
$r_{0_3} : 2z \to (z, here)$	$c_{0_3} = 1$
$r_{0_4} : s \to (s, in_1)$	$c_{0_4} = 1$
$r_{0_5} : s' \to (s, here)$	$c_{0_5} = 6 \cdot\ 10^{-2}$

Reaction	Constant
$r_{1_1} : s + u \to (p, out)$	$c_{1_1} = 1 \cdot 10^3$
$r_{1_2} : s + z \to (z + z', here)$	$c_{1_2} = 1$
$r_{1_3} : z' \to (z, out)$	$c_{1_3} = 1$

The simulation starts with a positive register value within V_1, represented by the u molecules; the system receives a sequence of `SUB` requests, due to the presence of s' molecules in V_0, transformed in s by the application of rule r_{0_5} and then sent to V_1 by rule r_{0_4}. Figure 5.25 shows the two execution phases: in the first phase the counter is decremented, as long as there are s' molecules available in V_0, and molecules p are produced in V_0. Afterwards, when the register counter reaches zero (all u molecules are consumed), only molecules z are produced in V_0.

This system is initialised with small quantities for molecular species, and this makes it fragile with respect to the inherent stochasticity, but our goal is to qualitatively show the required sharp change of behaviour occurring when the simulated register goes to zero.

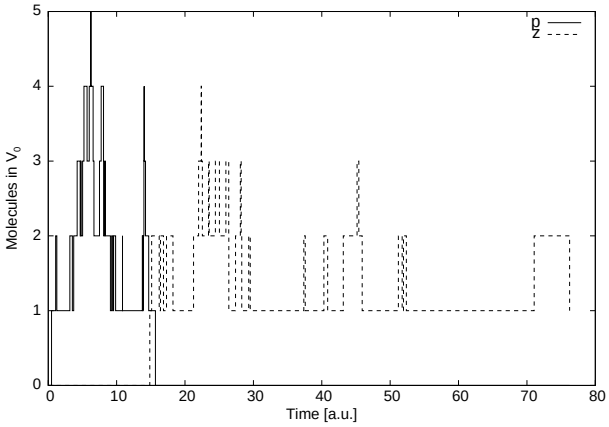


Figure 5.25: Plot of the dynamics of the SUB unit.

5.3.2.4 The SUBADD module

The SUB unit can be extended to perform both a `SUB` and an `ADD` instructions, according to the molecules received from the environment: s or a, respectively. The register value is stored inside a third volume, V_2, and it is represented by the amount of molecules u occurring inside of it. The rules shown in Table 5.7 are defined to perform both `SUB` and `ADD` operations, and the choice of molecular species avoid mixing them. In particular, rules $r_{0_1}, \ldots, r_{0_4}$, $r_{1_1}, \ldots, r_{1_6}$ and $r_{2_1}, \ldots, r_{2_3}$ define the SUB instruction (checking whether the register value is zero or not). The other rules are used to perform the ADD instruction.

The τ-DPP Υ_{SUBADD} implementing the SUBADD module is defined as

$$\Upsilon_{SUBADD} = (V_0, V_1, V_2, \mu, \mathcal{S}_\prime, \mathcal{S}_\infty, \mathcal{S}_\in, M_0, M_1, M_2, R_0, R_1, R_2, C_0, C_1, C_2),$$

where:

- V_0, V_1, V_2 are the volumes of the SUBADD module;
- μ is the nested membrane structure $[_0\ [_1\ [_2\ ]_2\ ]_1\ ]_0$;
- $\mathcal{S}_0 = \{l, l', s, z, m, n, p, k, k', a, A, o, q\}$, $\mathcal{S}_1 = \{s, p, z, a, A\}$ and $\mathcal{S}_2 = \{s, u, z,\ z', a, a'\}$ are the sets of molecular species;
- $M_0 = M_1 = \emptyset$ and $M_2 = \{u^{30}\}$ are the multiset occurring inside the membrane V_0, V_1 and V_2 at time $t = 0$.
- $R_0 = \{r_{0_1}, \ldots, r_{0_8}\}$, $R_1 = \{r_{1_1}, \ldots, r_{1_8}\}$, $R_2 = \{r_{2_1}, \ldots, r_{2_5}\}$ are the sets of rules defined in volumes V_0, V_1 and V_2 respectively, and reported in Table 5.7;

- $C_0 = \{c_{0_1}, \ldots, c_{0_8}\},\ C_1 = \{c_{1_1}, \ldots, c_{1_8}\},\ C_2 = \{c_{2_1}, \ldots, c_{2_5}\}$ are the sets of stochastic constants associated to the rules defined in V_0, V_1 and V_2, respectively, and reported in Table 5.7.

In Figure 5.26 (left), the simulation of a decrement instruction is shown. Inside the register there is a positive value (30 molecules u inside volume V_2); the system receives a `SUB` request, since l is sent to volume V_0. This request is processed by executing reaction r_{0_1} that produces l' and s; the latter is first sent to V_1 by means of reaction r_{0_2} and then to V_2 by applying reaction r_{1_1}. The value of the register is decreased by executing reaction r_{2_1} inside volume V_2, and a molecule p is sent to volume V_0 (passing through V_1) where the next instruction will be processed.

One the right side of Figure 5.26, the simulation of an increment instruction is shown. In this case, the label of the instruction is represented by the presence of a molecule k inside volume V_0. The molecule k is transformed into molecules k' (that is left inside volume V_0) and a (that is sent to volume V_1) by means of reaction r_{0_5}. a in then sent to volume V_2, by executing reaction r_{1_7}, where it is used to increment the value of the register (reaction r_{2_4}). The increment instruction inside volume V_2 produces a molecule a' that is transformed into A and sent to volume V_1 (reaction r_{2_5}). Finally A is sent to volume V_0 by means of reaction r_{1_8} where it is transformed into a molecule o by applying reaction r_{0_7}.

Table 5.7: Reactions for the SUBADD module (R_0 on the left, R_1 on the right and R_2 on the bottom).

Reaction	Constant
$r_{0_1} : l \to (l' + s, here)$	$c_{0_1} = 1$
$r_{0_2} : s \to (s, in_1)$	$c_{0_2} = 1$
$r_{0_3} : l' + z \to (n, here)$	$c_{0_3} = 1$
$r_{0_4} : l' + p \to (m, here)$	$c_{0_4} = 1$
$r_{0_5} : k \to (k' + a, in_1)$	$c_{0_5} = 1$
$r_{0_6} : a \to (a, in_1)$	$c_{0_6} = 1$
$r_{0_7} : k' + A \to (o, here)$	$c_{0_7} = 1$
$r_{0_8} : k' + A \to (q, here)$	$c_{0_8} = 1$

Reaction	Constant
$r_{1_1} : s \to (s, in_2)$	$c_{1_1} = 1$
$r_{1_2} : 2p \to (p, here)$	$c_{1_2} = 1$
$r_{1_3} : 2z \to (z, here)$	$c_{1_3} = 1$
$r_{1_4} : p + z \to (p, here)$	$c_{1_4} = 1$
$r_{1_5} : z \to (z, out)$	$c_{1_5} = 1$
$r_{1_6} : p \to (p, out)$	$c_{1_6} = 1$
$r_{1_7} : a \to (a, in_2)$	$c_{1_7} = 1$
$r_{1_8} : A \to (A, out)$	$c_{1_8} = 1$

Reaction	Constant
$r_{2_1} : s + u \to (p, out)$	$c_{2_1} = 1 \cdot 10^3$
$r_{2_2} : s + z \to (z + z', here)$	$c_{2_2} = 1$
$r_{2_3} : z' \to (z, out)$	$c_{2_3} = 1$
$r_{2_4} : a \to (u + a', here)$	$c_{2_4} = 1$
$r_{2_5} : a' \to (A, out)$	$c_{2_5} = 1$

5.3.3 Discussion and open problems

In this section we presented the framework of chemical computing for the implementation of logic gates and circuits, which can be modelled and simulated by using τ-DPP.

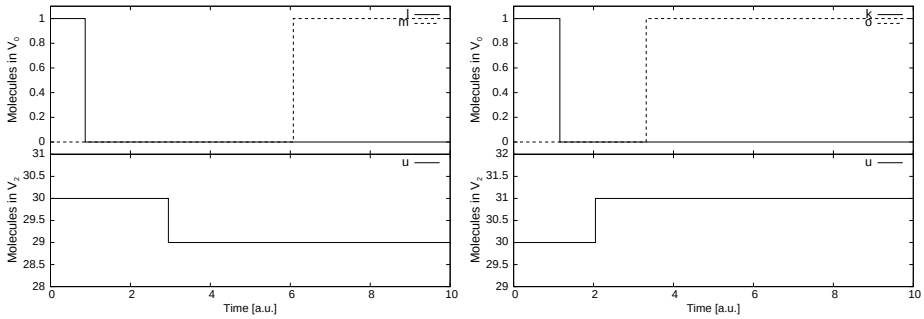


Figure 5.26: Plot of the dynamics of the SUBADD module performing a decrement instruction (left) and an increment instruction (right) on a register.

We first shown the results obtained from the simulation of simple logic gates, namely, the NAND and XOR. These gates represent the basic elements for the implementation of a "more complex" computing device. In fact, by composing a number of NAND or XOR logic gates, arranged according to a particular structure, it is possible to represent any boolean function.

Afterwards, we have shown how to model and simulate both the Fredkin gate and (an example of) Fredkin circuits with chemical reactions systems. The results presented in Section 5.3.2.2 show the behaviour of the system simulating the Fredkin gate; during the simulation the input values can be modified, producing a corresponding change in the output values. Moreover, the model of Fredkin gate has been used to build an example of Fredkin circuit (composed of three Fredkin gates, arranged in a tissue-like structure). Thanks to the modularity of the chemical system implementing the gate, every Fredkin circuit can be built by putting each gate inside a volume and adding as many communication rules as the lines connecting the gates within the circuit. Another aspect that is worth considering is that Fredkin circuits implemented as chemical systems are not "self reversible", that is, it is not possible to simulate both the forward and backward computations of the circuit without any modification of the system. Hence, in order to simulate the backward computations of a circuit, the flow of information described by communication rules needs to be reversed. This can be achieved, anyway, by constructing a similar system where the source and the target volumes of these rules are exchanged.

A possible improvement to the implementation of Fredkin gates and circuits hereby proposed includes the test of different sets of stochastic constants for the chemical reactions, in order to reduce the delay generated during the computation.

A different application of chemical computing within the τ-DPP framework consists in the simulation of the register machine instructions. We first

modelled the SUB instruction as a system composed by two membranes, which has a conditional behaviour, checking whether a register is zero or not. We then presented the results obtained by simulating the SUBADD module. Note that, by using these modules, we can outline the structure of a τ-DPP system simulating a complete register machine with just three levels of nested membranes, namely, the skin membrane, enclosing a number of "register" membranes structured like volume V_1 in SUBADD module.

The key idea to be developed, is to simulate the steps of the register machine by having inside the skin membrane the molecules representing the current instruction label.

For instance, if instruction l increments register r, then the rules would be defined to produce molecules a_r and send them to an internal membrane representing register r. That internal membrane will then produce molecules A_r, and a rule in the skin membrane would transform pairs of molecules $l + A_r$ into (non-deterministically chosen) molecules m, where m is one of the outcome labels specified by the `ADD` instruction being simulated. Additional details related to the halting of the computation need to be specified.

This approach to the implementation of complex systems leads to some open problems worth being studied. How does the passage from single simple components to complete universal devices, with the required connectivity, scale? It is well known that small universal register machines can be built, as shown in [102], but their τ-DPP implementation, and eventually their chemical system implementation have to be evaluated.

Moreover, the computational efficiency of these systems can be studied, for instance with respect to NP-complete problems such as SAT. Anyway, the usual trade-off between space and time in structural complexity perhaps has to be applied with negative results to τ-DPP, since objects could exponentially grow in polynomial time (by using rules like $p \to 2p$), but the space structure of volumes is fixed. Finally, also the stochasticity of τ-DPP has to be considered in the computational complexity study.

5.4 A study on the combined interplay between stochastic fluctuations and the number of flagella in bacterial chemotaxis

Chemotaxis is an efficient signal transduction pathway which allows bacterial cells to move directionally, in response to specific attractants or repellents occurring in their surroundings. The pathway consists of several transmembrane and cytoplasmic proteins acting as signal sensors and response regulators [97], which rule the reversal of the flagellar motor (governed by the phosphorylation and dephosphorylation of a key protein, CheY). This process induces a switch between running and tumbling movements, with a frequency that allows a temporal sampling (through random walks) of homogeneous environments. Anyway, in the presence of a gradient of attractants or repellents, the bacteria are able to respond quickly by reducing the frequency of flagellar reversal between clockwise and counterclockwise rotations, which cause a longer running motion in a biased direction. The frequency of switching is then reset to the random walk level if the concentration of the external ligands remains constant in time. At the molecular scale, this adaptation property is implemented by the coordinated action of methyltransferase and methylesterase proteins acting on the transmembrane receptors.

The genetic regulation and biochemical functions of the proteins involved in chemotaxis are well known, and several models have been proposed to study their complex interplay as well as the robustness of this system [5, 186, 138, 106, 110, 170]. In the model we present hereby, we consider very detailed protein–protein interactions for the chemotactic pathway in *E. coli*, in response to attractant molecules, which sum up to 62 biochemical reactions and 32 molecular species. The temporal evolution of the phosphorylated form of CheY (CheYp) is studied through stochastic simulations performed by means of the τ-DPP algorithm presented in Chapter 4. In particular, we investigate the CheYp dynamics under different conditions, such as the deletion of other proteins involved in the pathway, the addition of distinct amounts of external ligand, and the effect of different methylation states.

We then propose an analysis on the interplay between the stochastic fluctuations of CheYp and the number of cellular flagella – around half a dozen in *E. coli* – in the individual bacterium, to the aim of devising the mean time periods during which the cell either performs a running or a tumbling motion. Experimental observations show that the running motion requires all flagella to be simultaneously synchronized in the counterclockwise rotation, which occurs when CheYp is not interacting with the proteins regulating the flagellar motor; on the contrary, when at least one flagellum is not coordinated with the others, then the bacterium performs a tum-

bling movement. To distinguish between these two states, we assume that the cell is sensitive to a threshold level of CheYp, that is evaluated as the mean value of CheYp at steady state. Because of stochastic fluctuations, the amount of CheYp will randomly switch from below to above this value, thus reversing the rotation from counterclockwise to clockwise rotations of each flagellum. The original contribution presented here, is the link between synchronization of flagella and stochastic fluctuations of CheYp, as the core reason that stands at the basis of chemotactic motion. We have defined a procedure to identify the synchronization of rotations of all flagella, and we use it to compare the mean time intervals of running and tumbling motions – as well as of the adaptation times after ligand addition – according to a varying number of flagella.

5.4.1 The modeling of bacterial chemotaxis

In this section we present the chemotaxis signalling pathway and define the mechanistic model that describes the molecular interactions therein involved.

Bacterial chemotaxis Chemotaxis is a signal transduction pathway that allows swimming bacteria to perform biased movements in ever–changing environments, by efficiently sensing concentration gradients of beneficial or toxic chemicals in their immediate proximity. The binding of ligand molecules triggers an event cascade involving several transmembrane and cytoplasmic proteins, which lately affects the concentration of a pivotal response regulator, CheY. This protein rapidly diffuses inside the cell and interacts with the proteins of the flagellar motor, thus inducing clockwise (CW) and counterclockwise (CCW) rotation of each flagellum. When flagella are turning CW, they are uncoordinated and the bacterium performs a *tumbling* movement, while if they are all turning CCW, they form a bundle and get coordinated, thus allowing the cell to swim directionally (the so–called *running* movement). In a homogeneous environment, bacteria perform a temporal sampling of their surroundings by moving with a random walk, that is caused by a high switch frequency of the flagellar motors rotations, that alternate rapid tumblings with short runnings. In the presence of a ligand concentration gradient, instead, bacteria carry out directional swimming toward/against the attractants/repellents, by reducing the switch frequency of flagella rotations, that results in longer running movements. If the ligand concentration remains constant in time, then the switch frequency is reset to the prestimulus level, therefore realizing an *adaptation* of the chemotactic response to the change in ligand concentration. In what follows, we consider the chemosensory system of *E. coli* bacteria, in response to attractant chemicals.

The chemotactic pathway has been well characterized from a molecular point of view (see Figure 5.27). External signals are detected by transmembrane methyl–accepting proteins (MCPs), which are linked to cytoplasmic histidine protein kinases (CheA) by means of scaffold proteins (CheW). These three proteins constitute the sensor module (receptor complexes) of the whole pathway; each protein occurs as a dimer in every receptor complex. The role of CheA is to transduce the presence of an external ligand toward the inside of the cell, by phosphorylating two cytoplasmic proteins, called CheY and CheB. The transfer of the phosphoryl group to these proteins is more probable – that is, the activity of CheA is stronger – in absence of external ligands. CheY and CheB compete for the binding to CheA, but the phosphotransfer to CheY is faster than to CheB [202]; this fact assures that the proper chemotactic response can be generated before the process of adaptation occurs, as explained hereafter. CheY is the response regulator protein which, after being phosphorylated, interacts with the proteins FliM of the flagellar motors, inducing the CW rotation of the flagellum and the tumbling movements (FliM is a key component of the processes that stand *downstream* of the chemotaxis signalling, and therefore will not be explicitly included in our model; anyway, some considerations about its role within the model are discussed in Section 5.4.4). In presence of external ligands, the activity of CheA is reduced: the concentrations of phosphorylated CheY diminishes, its interaction with the flagellar motors is reduced, the CCW rotation is switched on, and bacteria can perform longer running movements. The termination of this signal transduction process is mediated by another cytoplasmic protein, CheZ, which acts as an allosteric activator of CheY dephosphorylation. Concurrently to the processes involving CheY, the chemosensory system possesses an adaptation response which depends on the methylation level of the receptors. Methylation reactions are modulated by the coordinated interplay between proteins CheR and CheB. Up to 4–6 methyl groups are constantly transferred to the cytoplasmic domain of MCPs by the constitutively active methyltransferases CheR. On the other side, the demethylation of MCPs occurs by means of the phosphorylated form of the methylesterase CheB. The methylation state of MCPs also intervene on the regulation of CheA: when MCPs are highly methylated, CheA is more active; when MCPs are unmethylated, the activity of CheA is reduced. In the latter case, also the concentrations of phosphorylated CheB diminishes, and this in turn lets the methylation state of MCPs increase, with a consequent renewed activity of CheA, and so on through a continuous feedback control. Therefore, the cell is able to adapt to environmental changes and return to the random walk sampling when the concentration gradient of the attractant remains constant in time. This feedback mechanism also allows bacteria to widen the range of ligand concentration to which they can respond, making them very sensible to low environmental variations.

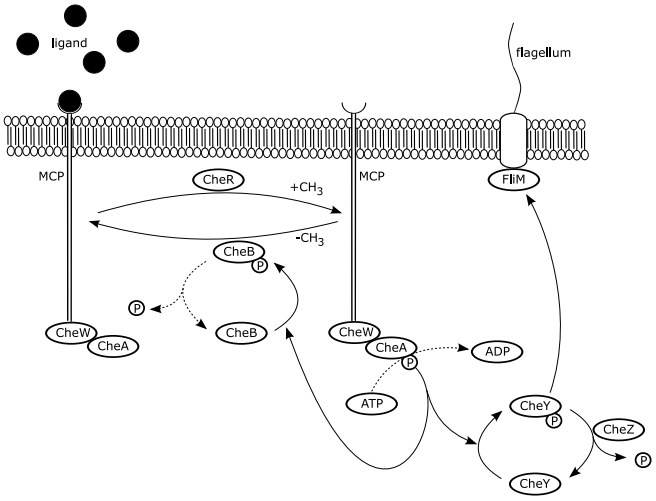


Figure 5.27: Signal transduction pathway in bacterial chemotaxis: solid arrows indicate enzyme–catalysed reactions, dashed arrows indicate autocatalysis; CH_3 denotes the methyl group, P the phosphoryl group (the dimensions of components are not scaled).

A mechanistic model For the modeling of the chemotaxis pathway, we have considered detailed protein–protein interactions which sum up to a total of 62 reactions and 32 molecular species [51]. The initial amounts – given as number of molecules – of the 7 elementary chemotactic proteins [184] are the following: 4000 dimers of MCPs; 4000 dimers of CheW; 4000 dimers of CheA; 17000 monomers of CheY; 12000 monomers of CheZ; 200 monomers of CheR; 1700 monomers of CheB (plus a constant amount of $1.2 \cdot 10^6$ ATP molecules that are needed for phosphorylation reactions). All other molecular species appearing in the model are obtained by mimicking the formation and dissociation of protein complexes, and by describing the phosphorylation/dephosphorylation of cytoplasmic proteins and the methylation/demethylation of MCPs, in both the conditions of presence and absence of external ligands.

Each reaction in the model is given in the form "reagents → products", where the notation X + Y is used to represent a molecular interaction between species X and Y, while X::Y denotes that X and Y are chemically bound in the formation of a complex (see Table 5.8). Note that only monomolecular o bimolecular reactions are here considered; the formation of complexes consisting of more than two species is performed stepwise. The phosphorylated form of species X, with X ∈ {CheA, CheB, CheY}, is denoted by Xp, while the methylation state of receptor MCP is denoted by MCP^m, for $m = 0, \ldots, 4$ (that is, five methylation states are considered).

The reactions describe the following molecular interactions:

Table 5.8: The 62 reactions of the model of bacterial chemotaxis. The values of the corresponding stochastic constants are: $c_1 = 0.1, c_2 = 0.01, c_3 = 0.1, c_4 = 0.02, c_5 = 0.325, c_6 = 0.29, c_7 = 0.165, c_8 = 0.05, c_9 = 0.0044, c_{10} = 0.0175, c_{11} = 0.0306, c_{12} = 0.035, c_{13} = 5.0 \cdot 10^{-7}, c_{14} = 7.0 \cdot 10^{-6}, c_{15} = 2.8 \cdot 10^{-5}, c_{16} = 5.0 \cdot 10^{-5}, c_{17} = 6.8 \cdot 10^{-5}, c_{18} = 5.0 \cdot 10^{-4}, c_{19} = 0.0035, c_{20} = 0.014, c_{21} = 0.025, c_{22} = 0.0336, c_{23} = 2.0 \cdot 10^{-4}, c_{24} = 0.0014, c_{25} = 0.0056, c_{26} = 0.01, c_{27} = 0.0135, c_{28} = 0.6, c_{29} = 0.8, c_{30} = 1.0, c_{31} = 1.2, c_{32} = 1.4, c_{33} = 15.0, c_{34} = 15.0, c_{35} = 15.0, c_{36} = 15.0, c_{37} = 15.0, c_{38} = 4.0 \cdot 10^{-4}, c_{39} = 3.75 \cdot 10^{-4}, c_{40} = 3.5 \cdot 10^{-4}, c_{41} = 2.125 \cdot 10^{-4}, c_{42} = 6.0 \cdot 10^{-4}, c_{43} = 0.0044, c_{44} = 0.0175, c_{45} = 0.0343, c_{46} = 1.0 \cdot 10^{-8}, c_{47} = 5.0 \cdot 10^{-7}, c_{48} = 7.0 \cdot 10^{-6}, c_{49} = 2.8 \cdot 10^{-5}, c_{50} = 5.0 \cdot 10^{-5}, c_{51} = 1.0 \cdot 10^{-5}, c_{52} = 5.0 \cdot 10^{-4}, c_{53} = 0.0035, c_{54} = 0.014, c_{55} = 0.03, c_{56} = 1.0 \cdot 10^{-5}, c_{57} = 2.0 \cdot 10^{-4}, c_{58} = 0.0014, c_{59} = 0.0056, c_{60} = 0.0112, c_{61} = 0.0080, c_{62} = 1.0$. The methylation states (m) corresponding to the reactions are: $m = 0$ for reactions $1-4$, $m = 0, \ldots, 3$ for reactions $5-8$ and $38-41$, $m = 1, \ldots, 4$ for reactions $9-12$ and $42-45$, $m = 0, \ldots, 4$ for reactions $13-37$ and $46-60$.

	Reagents	**Products**
1	2MCP^m + 2CheW	2MCP^m::2CheW
2	2MCP^m::2CheW	2MCP^m + 2CheW
3	2MCP^m::2CheW + 2CheA	2MCP^m::2CheW::2CheA
4	2MCP^m::2CheW::2CheA	2MCP^m::2CheW + 2CheA
5-8	2MCP^m::2CheW::2CheA + CheR	2MCP^{m+1}::2CheW::2CheA + CheR
9-12	2MCP^m::2CheW::2CheA + CheBp	2MCP^{m-1}::2CheW::2CheA + CheBp
13-17	2MCP^m::2CheW::2CheA + ATP	2MCP^m::2CheW::2CheAp
18-22	2MCP^m::2CheW::2CheAp + CheY	2MCP^m::2CheW::2CheA + CheYp
23-27	2MCP^m::2CheW::2CheAp + CheB	2MCP^m::2CheW::2CheA + CheBp
28-32	lig + 2MCP^m::2CheW::2CheA	lig::2MCP^m::2CheW::2CheA
33-37	lig::2MCP^m::2CheW::2CheA	lig + 2MCP^m::2CheW::2CheA
38-41	lig::2MCP^m::2CheW::2CheA + CheR	lig::2MCP^{m+1}::2CheW::2CheA + CheR
42-45	lig::2MCP^m::2CheW::2CheA + CheBp	lig::2MCP^{m-1}::2CheW::2CheA + CheBp
46-50	lig::2MCP^m::2CheW::2CheA + ATP	lig::2MCP^m::2CheW::2CheAp
51-55	lig::2MCP^m::2CheW::2CheAp + CheY	lig::2MCP^m::2CheW::2CheA + CheYp
56-60	lig::2MCP^m::2CheW::2CheAp + CheB	lig::2MCP^m::2CheW::2CheA + CheBp
61	CheYp + CheZ	CheY + CheZ
62	CheBp	CheB

- association of the three dimers (2MCP, 2CheW and 2CheA) constituting each ternary receptor complex (reactions 1–4);
- binding and unbinding of ligand molecules to the receptor complex in the five methylation states (reactions 28–32 and 33–37, respectively);
- methylation and demethylation of MCPs, in absence and in presence of ligand molecules (reactions 5–8 and 9–12, 38–41 and 42–45, respectively);
- autophosphorylation of CheA in the five methylation states of MCPs, in absence and in presence of ligand molecules (reactions 13–17 and

46–50, respectively);

- phosphotransfer to CheY in different methylation states of MCPs, in absence and in presence of ligand molecules (reactions 18–22 and 51–55, respectively);
- phosphotransfer to CheB in different methylation states of MCPs, in absence and in presence of ligand molecules (reactions 23–27 and 56–60, respectively);
- dephosphorylation of CheYp and CheBp (reactions 61–62).

According to literature, the ternary receptor complex $2MCP^m$::2CheW::-2CheA is assumed to be stable for the duration of the signal transduction process [185]; moreover, the synthesis and degradation rates of all chemotactic proteins are assumed to occur at a much slower scale than the chemotactic response (hence, the reactions corresponding to these processes have not been included in the model).

A stochastic constant is associated to each reaction, and it is needed to evaluate the probability of that reaction to occur, as explained in Section 1.1. The stochastic constants used for simulations are reported in the caption of Table 5.8 (all values are expressed in sec^{-1}). These values have been partly derived from literature [139], and partly tuned to account for the following biological features [109, 124]: (1) the binding affinity of the ligand is directly proportional to the methylation state of MCPs; (2) the ligand–receptor binding reactions occur at a faster rate with respect to phosphorylation and methylation/demethylation reactions; (3) the methylation and demethylation activities of CheR and CheBp are, respectively, inversely and directly proportional to the methylation state of MCPs; (4) the rate of phosphotransfer from CheA to CheY and CheB depends on the rate of autophosphorylation of CheA.

5.4.2 Stochastic simulations of chemotactic response regulator

In this section we show the results obtained for the chemotactic response regulator protein, the phosphorylated CheY (CheYp), by analysing the system under different conditions.

As already said, all the simulations have been performed using the τ-DPP algorithm, setting the error control parameter $\epsilon = 0.03$, as suggested in [26]. All the simulations have been performed using a Personal Computer with an Intel Core2 CPU (2.66 GHz) running Linux. The mean duration time for one run, for the simulation of the dynamics of CheYp over 3000 sec, is about 4–5 seconds (with the initial values of chemical amounts given in Section 5.4.1 and the constants reported in Table 5.8). All the figures

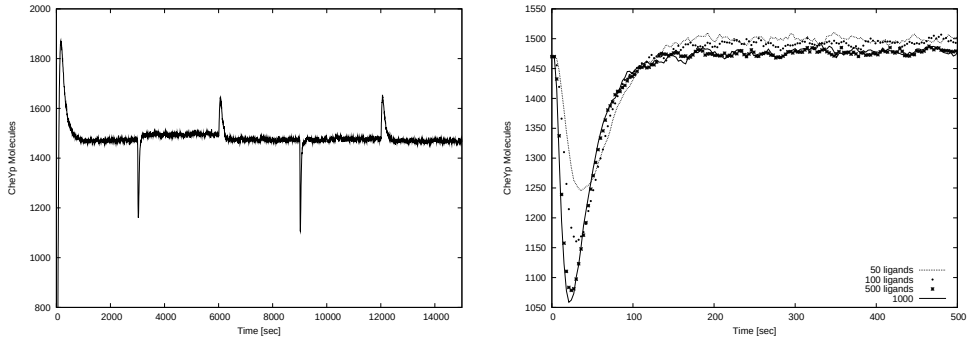


Figure 5.28: Dynamics of CheYp. Left: adaptation response to two consecutive stimuli. Right: comparison of transient and steady state response to different ligand amounts.

reported in the following, unless otherwise stated, represent the mean value over 50 independent runs of tau leaping, each one executed with the same initial conditions.

The dynamics of CheYp has been analysed by considering various conditions, such as the addition and removal of different ligand amounts, distinct methylation states of MCPs and deletion of other chemotactic proteins.

We start by reporting in Figure 5.28, left side, the response of the system to the addition of two consecutive amounts of external ligand: the first stimulus corresponds to a ligand amount of 100 molecules, added to the system at time $t = 3000$ sec and removed at time $t = 6000$ sec, while the second stimulus corresponds to a ligand amount of 500 molecules, added at time $t = 9000$ sec and removed at time $t = 12000$ sec. Note that, since the amount of CheYp is equal to 0 at the beginning of the simulation, its dynamics shows a marked increase which then reaches – due to the counteraction of CheZ, CheR and CheB – a steady state level. Starting from this level, the addition of the ligands has been simulated by changing its amount from 0 to 100 (500, respectively) molecules, thus mimicking the environmental situation where the bacterium encounters a different concentration of attractant molecules. Vice versa, the removal of the ligands has been simulated by putting the value of the ligand back to 0. In the time interval between the addition and the removal of each ligand stimulus, the amount of ligand molecules has been kept at the constant value of 100 and 500, respectively, thus mimicking the presence of an environmental homogeneous concentration. This has been done in order to test the adaptation capabilities of the system. In both cases, we can see that the system is able to respond to a step–increase of the ligands by achieving a sharp and fast decrease in CheYp (that is, the negative peaks at time instants $t = 3000$ and $t = 9000$ sec). Immediately after this transient, the amount of CheYp returns to a steady state value, which differs from the prestimulus level only

Table 5.9: Steady state values and minimal/maximal transient values of CheYp after addition and removal of distinct ligand amounts.

Ligand amount	**SS1**	**Min**	**SS2**	**Max**	**SS3**
50 molecules	1486.7	1245.4	1500.9	1626.0	1474.7
100 molecules	1486.7	1160.7	1495.1	1645.4	1474.3
500 molecules	1486.7	1078.4	1481.4	1653.2	1469.4
1000 molecules	1486.7	1058.6	1478.2	1665.8	1474.7

for a few tens of molecules, at most, according to the amount of added ligand. In this phase, the bacterium is returning to the prestimulus switching and thus to the random walk sampling of its surroundings. When the ligand is removed, CheYp shows another transient behavior, corresponding to a sharp and fast increase of its amount, that is in line with experimental observations (see [11, 170]). After this second transient, the amount of CheYp correctly returns to the prestimulus steady state level.

In Figure 5.28, right side, we compare the transients and steady states reached by CheYp after the addition of distinct ligand amounts. This figure shows that the response magnitude at steady state and the adaptation time of CheYp is only slightly sensitive to the ligand amount, being the relative differences less than a few tens of molecules and less than a few seconds, respectively. The mean values of the steady state of CheYp before the stimulus (SS1), after the ligand addition (SS2) and after the ligand removal (SS3) are reported in Table 5.9, together with the values of its minimal and maximal values immediately after the ligand addition and removal (Min and Max, respectively), for the four ligand amounts (50, 100, 500, 1000 molecules) considered in Figure 5.28.

In Figure 5.29 we show how the dynamics of CheYp changes when CheB is deleted from the system at time $t = 3000$ sec, in both conditions of absence of external ligands (left side) and of presence of 100 molecules of ligand (right side) added at time $t = 3000$ sec. CheB is the methylesterase that, once being phosphorylated by CheA, increases the methylation state of MCPs, thus keeping CheA more active. This, in turn, causes an increase in the amount of CheYp, which is evident from its new steady state level reached after CheB deletion, and also from its less negative transient decrease after ligand addition.

Similarly, in Figure 5.30 we show the dynamics of CheYp when either CheR (left side) or CheZ (right side) are deleted from the system at time $t = 3000$ sec, simultaneously to the addition of 100 ligand molecules (the temporal evolution of CheYp when no ligand is added is basically equivalent). When CheR is deleted, then its methyltransferase activity is silenced, the MCPs are no more methylated, and hence the amount of CheYp tends

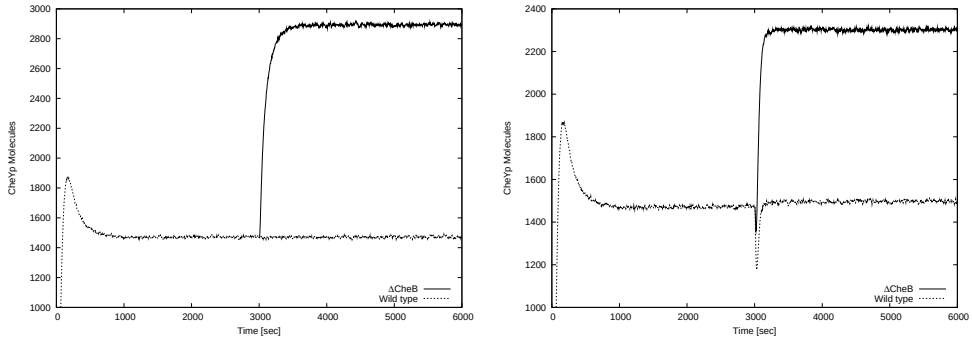


Figure 5.29: Comparison of dynamics of CheYp in normal condition and after deletion of CheB at $t = 3000$ sec, without ligand (left) and with simultaneous addition of 100 ligand molecules (right).

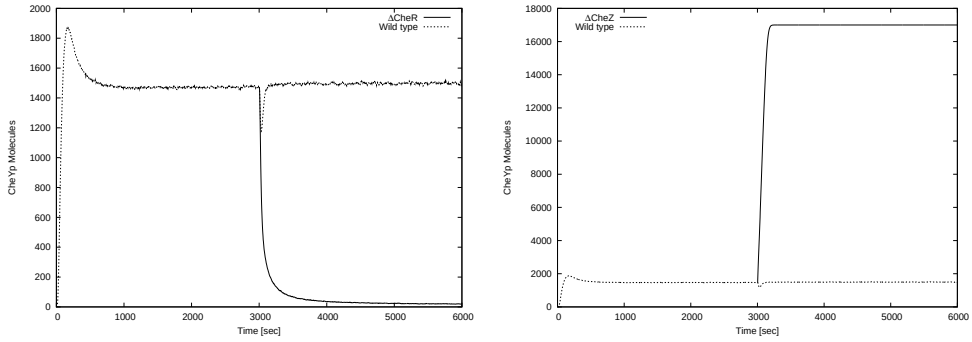


Figure 5.30: Comparison of dynamics of CheYp in normal condition and after deletion of CheR (left) and CheZ (right) at $t = 3000$ sec, with simultaneous addition of 100 ligand molecules.

to zero. On the contrary, when CheZ is deleted, then all CheY molecules always remain phosphorylated. For the sake of completeness, we have also simulated the dynamics of CheYp when either CheB, CheR or CheZ are deleted from the system since the time instant $t = 0$, in order to have a comparison about the initial temporal evolution of CheYp and the steady state levels it can reach. In these conditions, the model correctly simulates a very low production of CheYp when CheR is deleted, and an increased production (albeit with different magnitudes) when either CheB or CheZ are deleted (data not shown).

Finally, in Figure 5.31 we compare the dynamics of CheYp in response to the addition of 100 ligand molecules at $t = 3000$ sec, when only 3 (left side) or 2 (right side) methylation states of the receptors are allowed. In practice, this is achieved by initially putting to zero the values of the stochastic constants of methylation and demethylation reactions for level 4, and levels 3 and 4, respectively. In both cases, we see that the system is not able

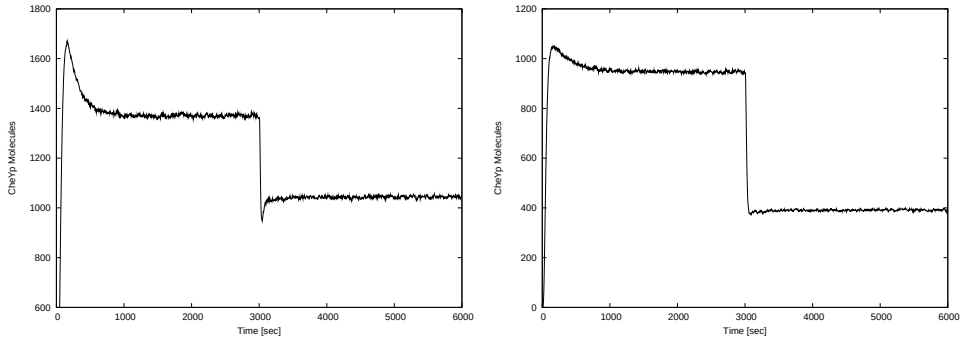


Figure 5.31: Dynamics of CheYp when only 3 (left) and 2 (right) methylation states are active.

to adapt, as the steady state level of CheYp reached after the addition of the ligand is substantially lower than the steady state when all methylation levels are activated.

5.4.3 The interplay between stochastic fluctuations and the number of bacterial flagella

In this section we analyse the interplay between stochastic fluctuations of CheYp and the number of flagella occurring on the cell, in order to outline the influence of synchronization of flagellar motors on the swimming behavior, and on the adaptation mechanism of the bacterium to the environmental changes. To this aim, we consider the dynamics of CheYp at steady state, as well as its transient step–decrease that takes place immediately after the chemotactic stimulus. In both cases, we are interested in devising the time periods during which the cell performs either a running or a tumbling motion. In particular we will assume that: (1) the time spent in alternating CW and CCW rotations during the steady state corresponds to the random walk sampling of the environment – where we expect more time spent in tumbling than in running motions; (2) the time required to return to the prestimulus level of CheYp (that is, the transient response immediately after the ligand addition) corresponds to the chemotactic adaptation time – where we expect a much longer and uninterrupted time interval of running motion with respect to the steady state condition.

As explained in Section 5.4.1, a running motion requires that all flagella are simultaneously synchronized in a CCW rotation – which occurs when CheYp is not interacting with the proteins FliM of the flagellar motors, that is, when its intracellular concentration diminishes with respect to a reference value. To distinguish between the CW and CCW rotations of a *single* flagellum, we assume that the flagellar motor switch is sensitive to a threshold level of CheYp, that is hereby evaluated as the mean value of CheYp at steady state (see also [138], where a similar approach of threshold–crossing

mechanism for motor switching was tested, albeit that work considered only single flagellum and did not propose any investigation on the simultaneous coordination of many flagella). When the amount of CheYp is below this threshold, each flagellum is rotating CCW, while when the amount of CheYp is above the threshold, each flagellum is rotating CW. In what follows, we make a one–to–one correspondence between the behavior of a single flagellum and one temporal evolution of CheYp generated by one run of the τ-DPP algorithm, that is, we consider a different and independent stochastic simulation for each and every flagellum (albeit starting from the same initial conditions). To determine the synchronization of all flagella, that will induce a running motion, we therefore need to identify the time instants when *all* flagella are rotating CCW, that is, to select the time intervals when *all* the temporal evolutions of CheYp are below the fixed threshold.

Formally, we proceed as follows. Let $n = 1, \ldots, 10$ be the number of flagella $f_1, \ldots, f_n$ whose influence we want to test, and let s_i, $i = 1, \ldots, n$, be the temporal series of CheYp (generated by a single τ-DPP run) associated to each f_i. For any fixed value of n, the total time of the simulation considered to generate the dynamics of CheYp is the same for all s_i. This simulation time, hereby denoted by Δt_{sim}, is chosen long enough to have statistical significance for the analysis performed below (e.g. $\Delta t_{sim} = 40000, 60000, 120000$ sec for $n = 1, 5, 10$, respectively). The threshold for CheYp is evaluated in the following way: we choose an initial time instant at the steady state level – distant enough from the step decrease of CheYp after ligand addition, i.e. 1000 sec afterwards – and then, starting from this instant and till the end of Δt_{sim}, we calculate the mean value $\mu_i =< s_i >$ for each s_i. Then, we define a common threshold μ for all flagella such that $\mu = \frac{1}{n} \sum_{i=1}^{n} \mu_i$. This threshold is considered as the reference value also for the portion of the CheYp dynamics corresponding to the transient decrease after ligand addition. In Figure 5.32, top panel on the left side, we show a part of Δt_{sim} over a single simulation of CheYp, where both the initial transient response and the stochastic fluctuations around the threshold are evidenced. For all the results discussed below, the different values of μ have been found to be approximatively equal to 1480 molecules.

The next step consists in detecting, for each f_i, the time intervals during which the amount of CheYp remains below μ, each one of these intervals corresponding to a CCW rotation time interval of that flagellum. Namely, for each s_i we identify the time intervals $\Delta t_{true} \subseteq \Delta t_{sim}$ such that $\Delta t_{true} = \{t \in \Delta t_{sim} \mid s_i(t) - \mu \leq 0\}$. Note that this simple mechanism of single threshold-crossing could be extended to consider more complex situations – e.g., a double threshold-crossing mode can be assumed – whereby one simply asks for analogous conditions to be satisfied. Similarly, for each s_i we can locate the complementary time intervals $\Delta t_{false} \subseteq \Delta t_{sim}$ such that $\Delta t_{false} = \{t \in \Delta t_{sim} \mid s_i(t) - \mu > 0\}$; these intervals correspond to the time that each flagellum f_i spends in a CW rotation. Stated in other terms, we

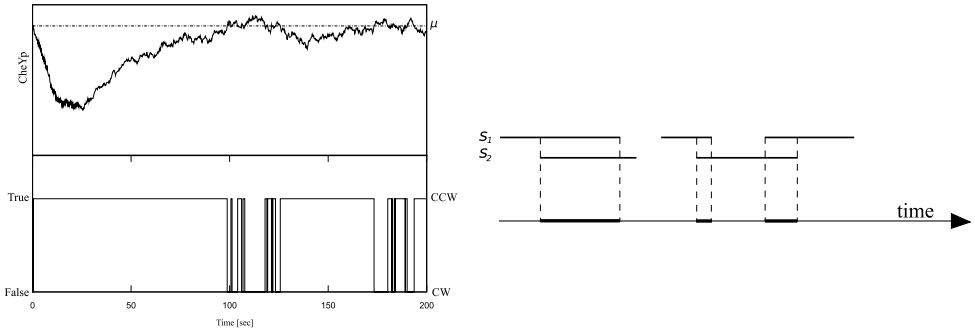


Figure 5.32: Threshold-crossing intervals in stochastic fluctuations of CheYp (left) and synchronization of running motion between 2 flagella (right).

can associate to each s_i a function $CCW_{s_i} : \Delta t_{sim} \rightarrow \{true, false\}$ defined as:

$$CCW_{s_i}(t) = \begin{cases} true & \text{if } s_i(t) - \mu \leq 0 \\ false & \text{otherwise} \end{cases}$$

In Figure 5.32, bottom panel on left side, we show the values of this function for the CheYp simulation given in the upper panel. As it can be seen at a glance, the transient response after ligand addition (when the amount of CheYp is initially below μ) corresponds to a longer and uninterrupted interval of CCW rotation.

Once that the set of all Δt_{true} intervals – or, equivalently, of function CCW_{s_i} – have been found out for each flagellum, we start the process of synchronization of these time intervals for the given number n of flagella. To this aim, let us define $\mathcal{T}^n_{sync} = \{t \in \Delta t_{sim} \mid CCW_{s_i}(t) = true \text{ for all } i = 1, \ldots, n\}$. $\mathcal{T}^n_{sync}$ is the set of all times during which *all* time series s_i are below the threshold μ, that is, the time intervals during which *all* flagella are rotating CCW. More precisely, we identify these intervals as the running motion of the bacterium, i.e. $\mathcal{T}^n_{sync}$ corresponds to the time of directional swimming – when all flagella are coordinated in a bundle. As an example, in Figure 5.32, right side, we show the set $\mathcal{T}^n_{sync}$ for $n = 2$. On the contrary, $\mathcal{T}^n_{unsync} = \Delta t_{sim} \setminus \mathcal{T}^n_{sync}$ corresponds to tumbling motion – when at least one flagellum (over the set of n flagella considered time by time) is rotating CW. Namely, $\mathcal{T}^n_{unsync} = \{t \in \Delta t_{sim} \mid \text{ there exists } i = 1, \ldots, n \text{ such that } CCW_{s_i}(t) = false\}$.

We are now interested in understanding if and how the time intervals within the set $\mathcal{T}^n_{sync}$ are influenced by the increase of n. For statistical significance, we have performed this analysis over a set of 10 distinct in silico experiments (each one corresponding to a cell with n flagella, with $n = 1, \ldots, 10$), and then we have evaluated the mean values of the following three parameters:

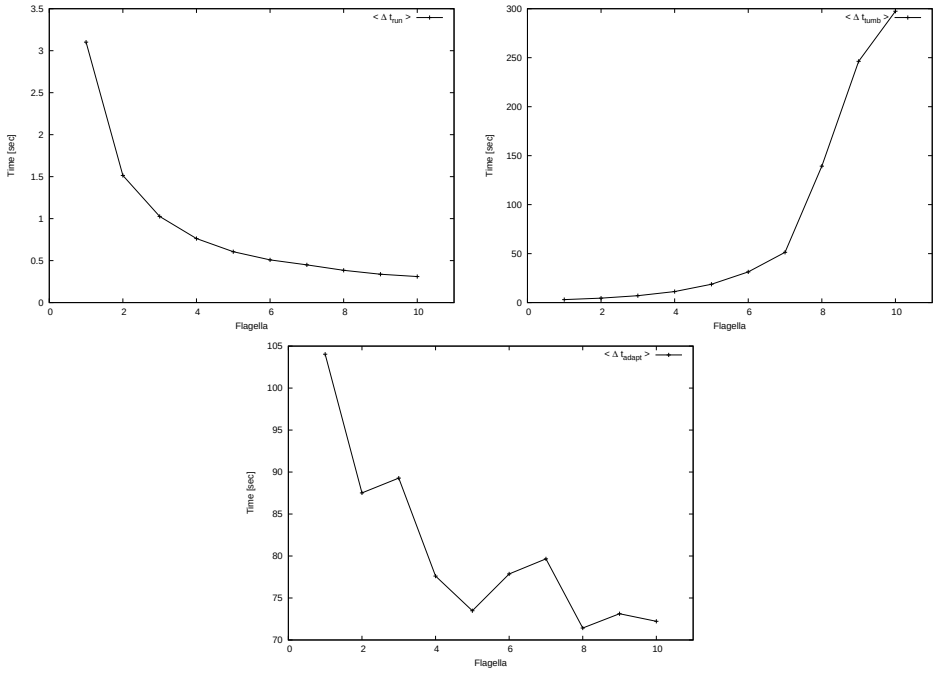


Figure 5.33: Variation of mean time values of running (left) and tumbling motions (right), and of adaptation time (bottom), with respect to the number of flagella.

1. the time intervals corresponding to a running motion of the bacterium ($< \Delta t_{run} >$), when all flagella are rotating CCW (that is, when all time series s_i are below μ);

2. the time intervals corresponding to a tumbling motion of the bacterium ($< \Delta t_{tumb} >$), when at least one flagellum over the n flagella is rotating CW (that is, when at least one time series s_i is above μ);

3. the time intervals corresponding to the transient decrease of CheYp after ligand addition ($< \Delta t_{adapt} >$), that is, the adaptation time during which the bacterium is performing a longer running motion.

The results for $< \Delta t_{run} >$ are reported in Figure 5.33, left side, where we can see that the mean time intervals of running motion are very short, and their values decrease in a (qualitative) exponential way as the number n of flagella increases. On the opposite side, the results for $< \Delta t_{tumb} >$ evidence a (qualitative) exponential increase with respect to n, as reported in Figure 5.33, right side. As reference, the precise values of the mean running and tumbling time intervals are given in Table 5.10, together with their ratio, for three values of n. The running–to–tumbling ratio, which decreases as n

Table 5.10: Values of mean time intervals for running, tumbling and adaptation.

n	$\langle \Delta t_{run} \rangle$ (sec)	$\langle \Delta t_{tumb} \rangle$ (sec)	$\langle \Delta t_{run} \rangle / \langle \Delta t_{tumb} \rangle$	$\langle \Delta t_{adapt} \rangle$ (sec)
1	3.102	3.062	1.013	104.0
5	0.606	18.73	0.032	73.48
10	0.310	297.4	0.001	72.22

increases, highlights the relevance of the number of flagella and the necessity of their synchronization with respect to the chemotactic behavior of the bacterium. That is, we see that for $n = 1$ the time spent in running or tumbling motions is approximatively equivalent, but if coordination among many flagella ($n = 10$) has to take place, then the running motions are highly reduced with respect to tumbling motions, which is in agreement with biological expectations.

The results for $< \Delta t_{adapt} >$ are reported in Figure 5.33, bottom, and in Table 5.10. In this case, it is not possible to recognize a simple function for the curve progress, and we see that the variation of the time intervals is within a range of a few tens of seconds. Once more, this result seems to be in agreement with biological expectations, as the response of the bacterium to an environmental change (i.e. the presence of ligands) should not be strictly dependent on the number of flagella that are present on its surface, otherwise the chemotactic pathway would not guarantee an appropriate adaptation mechanism, independently from the variation of the number of flagella among individuals in a population of cells.

5.4.4 Discussion

We have proposed a very detailed mechanistic model of the bacterial chemotaxis pathway that takes into account all proteins, and their respective interactions, involved in both signalling and response. In particular, all post-translational modifications of proteins, such as methylation and phosphorylation, have been considered because of their relevant roles in the feedback control mechanisms governing this pathway. By exploiting the τ-DPP algorithm, we have investigated the dynamics of the pivotal protein involved in chemotaxis, CheYp, under different conditions, such as the deletion of other chemotactic proteins, the addition of distinct amounts of external ligand, the effect of different methylation states. Then, we have investigated the possible influence of stochastic fluctuations of CheYp with respect to an increasing number of flagella in the individual bacterium. Namely, we have defined a procedure to identify the synchronization of CCW rotations of all flagella, and then we have compared the mean time intervals of running and tumbling motions of the cell, as well as of adaptation times to

ligand addition, according to the different numbers of flagella. We have shown that the running–to–tumbling ratio highlights the relevance of the number of flagella, and the necessity of their synchronization with respect to the chemotactic behavior of the bacterium. Moreover, we have shown that the adaptation time does not seem to be strongly influenced by the varying number of flagella in distinct individual cells.

Concerning the analysis of the interplay between CheYp fluctuations and the number of flagella, another aspect could be investigated, such as the influence of the amount of external ligands on the adaptation times (in Section 5.4.3, in fact, only the addition of 1000 ligands was used to execute all simulations). We will consider this matter in a forthcoming extension of this work. In addition, other relevant biological aspects of chemotaxis, that stand downstream of the signalling process, might represent valuable points to be included in a future study. For instance, it is known that each flagellar motor switch–complex is constituted by a group of proteins, called FliM, FliN, FliG (assembled in a ring), and by the torque–generating complexes, called MotA and MotB. In *E. coli*, a typical flagellar ring contains 34 copies of FliM, each of which can bind one copy of CheYp. In [53] it is suggested that binding of CheYp to FliM modifies the displacement of protein FliG, which directly interacts with the Mot complexes and therefore modulates the switch state of the flagellum. Moreover, flagellar motor switching has been found to be highly sensitive to the concentration of CheYp (having a Hill coefficient ≈ 10), though the binding of CheYp to FliM has a low level of cooperativity (Hill coefficient ≈ 1). In [53], the hypothesis that CheYp can interact more favourably with the FliM displaced in the CW orientation, than those in the CCW orientation, is put forward. In [181], in addition, the following mechanism is considered for the control of flagellar motor by means of CheYp: the number of CheYp molecules bound to FliM determines the probability of CW or CCW rotation, while the switch is thrown by thermal fluctuations. In other words, CheYp only changes the stabilities of the two rotational states, by shifting the energy level of CCW–state up and of CW–state down, by a magnitude that is directly proportional to the number of bound molecules. Therefore, interesting questions related to stochastic fluctuations of CheYp, that might be coupled with the investigation on the number of cellular flagella, are: how many FliM proteins at each flagellar switch have to be occupied by CheYp in order to generate the CW rotation [21]? What is the corresponding probability of throwing the reversal switch? Can a double–threshold crossing mechanism [21] be more suitable to effectively detect the CW and CCW rotational states?

Nonetheless, other related features might be relevant points for a further extension of this work. For instance, the model does not take into account the diffusion processes of molecules inside the volume where reactions occur. Though, the gradient of CheYp that can be present inside the cytoplasm – due to the diffusion from the area of its phosphorylation (close to chemotac-

tic receptors) to the area of its activity (around the flagellar motors) – can be a significant aspect in chemotaxis, together with the localization of CheZ (that controls the dephosphorylation of CheYp) and of the flagella around the cell [110].

We believe that the definition of detailed mechanistic models, like the one proposed here for chemotaxis, and the use of efficient tools for the analysis of stochastic processes in individual cells, can be a good benchmark to investigate the combined roles of many interplaying biological factors. With this perspective, the development of formal methods specifically devised for the analysis of properties of stochastic systems represents indeed a major hot topic research in biological modeling.

Chapter 6

The role of parameters in chemical and biological systems

The modelling of biological systems requires the knowledge of many numerical factors, like the concentrations of molecular species or reaction rates, which represent an indispensable quantitative information to perform computational investigations of the system behavior. Unfortunately, the experimental values related to molecular quantities and rate constants are often not available or inaccurate [173]. This lack of information has led to the development of different techniques, based on local or global optimisation methods, in order to estimate the unknown parameters [137].

In addition, once that the calibration of the parameters of a systems has been done, it is interesting to study and analyse the effect of the combination of different values of the parameters by means of ad hoc techniques (e.g. parameter sweep) that are suitable to the exploration of these very large and complex research spaces. Parameter sweep can be realised by executing a very large number of simulations, each one corresponding to a different parametrisation of the analysed system, in order to test its robustness and to find its crucial points.

In this chapter, we will first present the issue of *parameter estimation* for biochemical systems (Section 6.1), together with a brief description of the optimisation techniques that are already present in literature for this scope. Afterwards, we recall the basic definition of *parameter sweep application* (Section 6.2) emphasising its suitability for the exploration of the parameters space of biological systems.

Both parameter estimation and parameter sweep require the definition of an appropriate *fitness function*, in order to quantify the quality of a particular set of parameters. To be more precise, this function is used to measure the "difference" between the estimated dynamics, that is, the behavior ob-

tained through a simulation using the set of parameters under investigation, and a target dynamics, which represents a known behaviour of the system. The fitness function used hereby, will be described in Section 6.3.

In Section 6.4, we show how we can tackle the problem of estimating the unknown parameters of stochastic biochemical systems by means of two optimization heuristics, genetic algorithms and particle swarm optimization [15]. Their performances are tested and compared on two basic kinetics schemes: the Michaelis–Menten equation and the Brussellator. The experimental results suggest that particle swarm optimization is a suitable method for this problem. The set of parameters estimated by particle swarm optimization allows us to reliably reconstruct the dynamics of the Michaelis–Menten system and of the Brussellator in the oscillating regime.

Finally, in Section 6.5 we present a possible implementation of parameter sweep application obtained by distributing a number of stochastic simulations on the EGEE project grid platform [140]. As a case study, we present a parameter sweep application for a stochastic model of bacterial chemotaxis composed of 59 reactions and 31 chemical species. In order to analyse the effect of parameter variation over the dynamics of this system, we have performed a large number of simulations to explore the 59-dimensional space of its stochastic constants, and considered the temporal evolution of a pivotal protein as the reference dynamics. The performance and the results obtained from the different parameter sweep applications executed are finally discussed.

6.1 Parameter estimation of biochemical systems

In computational investigation of biochemical systems, the first step consists in understanding the system structure (that is, the molecular species and compartments involved in the system, their localisation and topology, etc.) and in developing a model to describe the interactions among the system components. The second stage corresponds to the study of the dynamics of the system, under different conditions, in order to yield useful predictions about any unknown interactions which can take place within the system. To achieve these tasks, it is important to identify the system structure as well as the set of parameters, because both are needed to analyse the behavior of such systems. The final stage consists in comparing the obtained computational results with the experimental results, in order to validate the developed model [101].

One of the most challenging step in this process, which is usually due to the non–linearity of these systems, is the estimation of the large set of numerical parameters that they contain. These parameters are: molecular amounts, binding constants, transcription rates, translation rates, chemical reaction rates, degradation rates, diffusion rates, etc. Except for special cases where the experimental values of these factor can be found in literature, they are often not available or inaccurate, since carrying out their measurements *in vivo* can be tangling or even impossible [173]. In a few cases, the values of some parameters of a given system can be estimated either from *in vitro* experiments (by fitting the dynamics derived through equations based on mass-action law against the concentration time series that result from these measurements), or by assuming some analogies with similar processes or organisms for which more experimental data are available. In general, it is exactly the lack and the inaccuracy of these information that bring about the problem of assigning the correct values to all parameters, in order to reproduce the expected dynamics in the best possible way.

The parameter estimation issue, also called *inverse problem* [126], consists in the calibration of the unknown system's parameters by means of optimisation techniques. It is called "inverse" because it is the opposite of the "forward" problem of the simulation of a system's dynamics. The inverse problem requires methods that can combine simulation techniques with optimization algorithms, capable of searching global optima in large (and complex) multi–dimensional spaces.

In general, the calibration of the parameters values can be tackled according to the following simple procedure [127]:

1. Simulate the model with the given set of parameters;

2. Compare the known experimental time course with the simulation outcome;

3. If the difference between these two dynamics is "small enough", then stop, otherwise adjust the parameters of the model;

4. Go back to step 1.

Step 1 of the procedure consists in the simulation of the model, which is the "inverse" problem of optimisation. During this step, given a set of fixed parameters, the dynamics of the system is described. Note that, usually, in the first iteration of this procedure, the initial values for the parameters are randomly chosen according to the chosen optimisation technique. Afterwards, the experimental data (if available) or a generic "reference dynamics" is compared to the results of the simulation in order to quantify the quality of the set of parameters previously assigned to the model. This task is accomplished by computing the value of a fitness function. During step 3, the value of the fitness function is checked in order to decide if the set of parameters needs further refinements, or if it is acceptable according to some specified criterion. In the first case, the values of the parameters are adjusted using an optimisation technique, such as e.g. hill climbing, simulated annealing, genetic algorithms or particle swarm optimizer, and another iteration of the procedure is executed. In the second case, the procedure terminates.

In general, the inverse problem is complicated by the non–linearity of the system's dynamics which is very often multimodal (i.e. nonconvex) [127]. The first attempt to tackle this kind of optimisation problems, the so–called *multistart strategy*, has been presented in [83]. This method consists in the repeated application of local methods, starting from a number of different initial sets of parameters for the analysed model. However, this approach is not suitable for real applications because of its inefficiency; as a matter of fact, when many starting points are used, the same minimum is determined several times. Moreover, the execution of a large number of instances of this method is very time consuming.

The limitations of local methods lead to the application of global optimisation methods in order to improve the efficiency and the robustness. In particular, in [127], the parameter estimation of the mechanism of irreversible inhibition of HIV proteinase has been presented, and the best results have been obtained by using the *simulated annealing* method. However, also in this case the main drawback is represented by the computational time required by the execution of this procedure.

Several different stochastic optimisation methods, applied to the parameter estimation of a large three-step pathway, that is, a case study consisting in the optimisation of 36 kinetic parameters, have been compared in [125]. The best performing method found was the *evolutionary programming*, since the solution obtained by the application of this procedure allowed to describe the correct dynamics of the analysed system, while other methods failed. More recently, in [137], several global optimisation methods have

been applied for the calibration of 36 parameters of a benchmark model. The most suitable method for the solution of the inverse problem, among those considered by the authors, was the *evolution strategy using stochastic ranking* (SRES), presented in [178]. SRES is an evolutionary optimisation algorithm that uses stochastic ranking as constraint handling technique, a method that adjusts the balance between the objective and the penalty functions automatically during the evolutionary search.

The main disadvantage of these global optimisation methods is the time required to perform a computation; however, it is well known that the methods cited above can be easily parallelised, thus reducing the computational time needed to obtain good results.

The method for the parameter estimation that we will propose in the following sections exploits two population based optimisation heuristic: genetic algorithms and particle swarm optimizer. These methods are suitable for the optimisation of sets of real values, like the parameters of biochemical systems, and their efficiency in the exploration of the parameter space has been proved by many applications. Moreover, our approach for the parameter estimation, like those presented above, can be parallelised, for instance, by distributing the computation of the fitness functions (which is usually the most time consuming task of the procedure) on a cluster.

6.2 Parameter sweep application

The behaviour of biochemical models is mainly influenced by the initial conditions, i.e. the molecular quantities, and by the rate constants values assigned to the chemical reactions. Real models are usually composed of a large number of chemical species which interact through many chemical reactions, hence, the space constituted by all possible combinations of values of the parameters has a high number of dimensions. Therefore, the exploration of such kind of spaces, to the aim of quantifying the influence of the parameter values on the system dynamics, is a hard task.

There exist many different techniques suitable for this kind of analysis. For instance, *steady state analysis* concerns the identification of points in the space of reachable states, where the dynamical system is fixed (e.g. where the behaviour of the system is constant over time); *bifurcation analysis* studies the qualitative variation of the dynamics (e.g. transition from oscillating to non oscillating regime) as a consequence of the variation of the parameters; *sensitivity analysis* relates the uncertainty of the input of a model (i.e. variations in parameters and initial conditions) to its output (namely, the resulting behaviour); and *parameter sweep application* (PSA) explore the parameters space of a system by means of independent experiments.

Among the techniques cited here, PSA represents one of the simplest method to execute, and it is very easy to run it on a parallel architecture.

The features of PSA makes it suitable to analyse the complex and non–convex parameters space of chemical and biological systems, where a very large number of simulations is needed. The aim of PSA consists in comparing the dynamics resulting from a particular setting with respect to the "reference dynamics", obtained by using the correct parametrisation. So doing, it is possible to test the robustness or the fragilities of a model.

To be more precise, PSA is an application that composes high–throughput computing applications for processing on parallel architectures. This application is a combination of task and data parallel models. Applications formulated by means of PSA contain large number of independent jobs operating on different data sets. A range of scenarios and parameters to be explored can be applied to the input values to generate different data sets. This application is executed by processing N independent jobs (each with the same task specification, but with a different data set) on M distributed computers (where N is, typically, much larger than M). Fortunately, this high–throughput parametric computing model is simple, yet powerful enough to formulate distributed application ranging in many different areas.

PSA occurs frequently in scientific computation across a broad range of disciplines. For example, this method is used in bio-medical [151], bioinformatics [91], bio-physics applications [158], data mining [187] and many other scientific domains.

6.3 Fitness function

The parameter estimation methods and PSA require a function able to quantify the quality of a particular set of parameters. This *fitness function* is used to measure the "difference" between a given experimental outcome (target dynamics), which represents the observed behaviour of the analysed system, and the dynamics obtained by means of simulations. Stated in other words, the "reference dynamics" is the behaviour that has to be reproduced by calibrating the parameters (in the case of parameters estimation issue), or the behaviour with respect to the dynamics obtained by perturbing one (or more) parameters is compared (in the case of PSA).

Working in the field of stochastic modelling and simulation, the fitness definition is based on the idea that we have to compare the target dynamics with the estimated dynamics that are generated by using a stochastic simulation algorithm (which will run using a particular set of parameter values time by time). Therefore, we have to manage some troublesome properties, hereby discussed, that are inherent to stochastic simulations. We start by introducing the definition of fitness function that will be used in the rest of this chapter. Briefly, the fitness of each individual will be evaluated by

calculating the *area* between the target dynamics[1] (TD) and the estimated dynamics (ED) of each molecular species.

To be more precise, the fitness function is defined as follows. Let $L \subseteq \{1, \ldots, M\}$ be the set of molecular species whose dynamics is assumed to be experimentally known, t_0 and t_N the initial and final time instants of the given TD. We denote by $x'_l(t_0), \ldots, x'_l(t_N)$ the TD time series of species l, $l \in L$, and by $x_l(\tau_0), \ldots, x_l(\tau_{\widetilde{N}})$ its ED time series, where $\tau_0 = t_0$ and $\tau_{\widetilde{N}} \cong t_N$. Note that, in general, all the time instants (except the initial one) of the ED series will be distinct from those sampled in the TD series, since the two methods use different time samplings and, above all, each simulation performed by tau leaping generates a different time series (hence, it does not correspond to the constant-step temporal sampling of TD). In addition, the time interval between any couple of consecutive time instants of the ED series will be generally distinct from every other time interval in the same series. As the evaluation of the fitness function requires to determine the difference between the TD and the ED, we need to pick up couples of values, one in the TD series and the other one in the ED series, that correspond to an identical time instant. Therefore, for each simulated time instant τ_i, $i = 0, \ldots, \widetilde{N}$ (drawn as the "evaluated" time series in Figure 6.1), we consider the two consecutive time instants in the TD series t_j, t_{j+1}, $j = 0, \ldots, N-1$ (drawn as the "target" time series in Figure 6.1), which satisfy the following conditions: (1) $t_j \leq \tau_i \leq t_{j+1}$; (2) there exist no other time instants t'_j, t'_{j+1}, $j = 0, \ldots, N-1$, such that $t_j \leq t'_j \leq \tau_i \leq t'_{j+1} \leq t_{j+1}$. Then, by performing a linear interpolation between the given TD series values $x'_l(t_j)$ and $x'_l(t_{j+1})$, we derive the element $x'_l(\tau_i)$ (drawn as the "interpolated" time series in Figure 6.1) in the TD series corresponding to the element $x_l(\tau_i)$ in the ED series, and we evaluate their distance, $|x'_l(\tau_i) - x_l(\tau_i)|$. We then compute the fitness f_z (for each independent execution z of the tau leaping algorithm) by summing up, for each simulated time instant τ_i and for each molecular species $l \in L$, the areas of the trapezoids (thick lines in Fig. 6.1) having as basis the distances $|x'_l(\tau_i) - x_l(\tau_i)|$ and $|x'_l(\tau_{i+1}) - x_l(\tau_{i+1})|$, and as height the length of the time interval $[\tau_i, \tau_{i+1}]$, that is: $f_z = \sum_{i=1}^{\widetilde{N}-1} \sum_{l \in L} \frac{1}{2}(|x'_l(\tau_i) - x_l(\tau_i)| + |x'_l(\tau_{i+1}) - x_l(\tau_{i+1})|)(\tau_{i+1} - \tau_i)$. Finally, the fitness function is defined as $f = \frac{1}{Z} \sum_{z=1}^{Z} f_z$, where Z is the total number of independent runs of the tau leaping algorithm executed by using the same set of parameters.

There exist other methods to compute the value of the fitness function; for instance, by using the root mean square the "difference" between the TD and the ED can be measured. Clearly, in order to calculate the correct value for the fitness function, we need to interpolate the ED to obtain a regular sampling. However, this strategy is much more time consuming with respect

[1] In the examples proposed in the following sections, pseudo-experimental TD are generated by means of an ODEs solver, or as the average dynamics obtained as the "mean" behavior of a set of stochastic simulations.

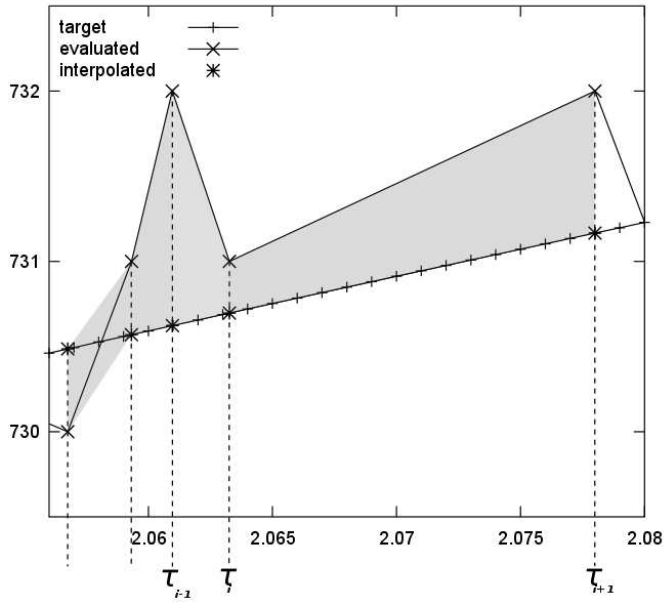


Figure 6.1: Area evaluation between TD and ED series for the fitness function.

to the computation of the area between the two analysed curves, since the number of operations needed to linearly interpolate the points of the ED is higher with the root mean square than that needed to compute the areas of the trapezoids denoted by the two time series.

6.4 A comparison of GAs and PSO for parameter estimation in stochastic biochemical systems

Optimization methods can be used to tackle the calibration problem of *parameter estimation* of biochemical systems by minimizing a cost function (e.g. a distance measure) which quantitatively defines how good is the system behavior using the predicted values, with respect to the experimental dynamics. Several global optimization techniques (see [173, 137] and references therein) have already been adopted for parameter estimation of biochemical and biological systems. A peculiarity of our approach, with respect to other methods previously investigated for this problem, is that it embeds the (forward) problem of performing *stochastic* simulations of a system dynamics, into the inverse problem of estimating its unknown parameters. This leads to the development of a method that exploits the outcome of stochastic simulation algorithms to effectively compute the fitness function value used by the optimization techniques here investigated. In particular, we compare the performances of genetic algorithms (GAs) and particle swarm optimization (PSO) for the parameter estimation of two simple biochemical schemes which are well representative of the dynamics of many other biological systems: a basic catalytic kinetics (the Michaelis-Menten system), a sustained oscillating behavior and a damped oscillating behavior (both based on the Belousov-Zhabotinskii reaction).

The choice of these two particular optimisation techniques is motivated by the fact that parameter estimation is an example of dynamic optimization problems, meaning that the fitness function may change (more precisely, each individual can have slightly different fitness values each time it is evaluated) and a number of contributions exist about the use of GAs and PSO for this kind of problems (see, e.g., [115, 95]).

In the following sections we will recall some basic notions of biochemical reactions, and describe the two simple systems (Michaelis-Menten reaction and the Brussellator) that we will consider for parameter estimation. Afterwards, we describe the versions of GAs and PSO that we have used. The experimental results are finally presented together with further applications of these methods and some open problems.

6.4.1 Systems of biochemical reactions

Any generic (bio)chemical reaction can be written in the form $r : \sum_{i=1}^{N_1} \alpha_i R_i \rightarrow \sum_{j=1}^{N_2} \beta_j P_j$, where $R_1, \ldots, R_{N_1}$ are distinct reactant molecular species and $P_1, \ldots, P_{N_2}$ are distinct products, for some $N_1, N_2 \geq 0$. The non-negative integers α_i, β_j are the stoichiometric coefficients of the reaction r; they specify how many molecules of each reagent species are necessary to trigger the reaction, and how many product molecules are formed after the reaction has occurred. Each reaction is also characterized by a numerical factor,

called the rate constant, which determines – together with the amount of reagents – the rate, or velocity, of the reaction itself. The kinetics of a reaction is influenced by many factors, such as temperature, pressure, amounts of reactants, molecular crowding, etc. As a consequence, the experimental determination of rate constants for a given biochemical system is not a trivial task, and the situation gets even harder when considering more complex biological systems, like metabolic pathways or cellular processes in general. For further notions about biochemical reactions and chemical kinetics we refer to [39, 146].

Michaelis-Menten system. The first chemical system we consider, the *Michaelis-Menten* (MM) kinetics, describes the catalytic transformation of one-reacting substrate (denoted by S) into a final product (P) mediated by an enzyme (E), passing through the (relatively fast) reversible formation of the enzyme-substrate intermediate complex (ES). The role of E is to lower the energy required by S for its interconversion to P; the enzymatic kinetics assumes that S and E are in a fast equilibrium with the complex they form, which then dissociates to yield the product while releasing the enzyme free. The set of chemical reactions corresponding to MM is: $E + S \xrightarrow{c_1} ES, ES \xrightarrow{c_2} E + S, ES \xrightarrow{c_3} E + P$, where c_1, c_2, c_3 are the stochastic constants that encompass the physical and chemical properties of the reaction. The initial amounts of the substrate and the enzyme used to generate the target dynamics for our problem of parameter estimation are $S = 1000$, $E = 750$ molecules, while the stochastic constant values (that we want to estimate with GAs and PSO) are $c_1 = 2.5 \cdot 10^{-3}, c_2 = 0.1, c_3 = 5$. The dynamics corresponding to this set of parameters is shown in Figure 6.5 (top left graph, with solid lines).

Brussellator system. The second chemical system we consider, called *Brussellator* [199], is a simplified scheme for the Belousov-Zhabotinskii reaction, a family of inorganic redox reaction systems that exhibit macroscopic temporal oscillations and spatial patterns formation. This theoretical scheme is recognized as the prototype of nonlinear oscillating (open and well-stirred) systems, proving the significance and variety of both spatial organizations and complex rhythms occurring in many biological systems. Here, we give a description of the Brussellator that slightly differs from the original one: with respect to the formulation given in [199], we leave out the presence of two products (since they are not directly involved in the formation of the oscillating limit cycle), and we consider the following set of reactions: $A \xrightarrow{c_1} X, B + X \xrightarrow{c_2} Y, 2X + Y \xrightarrow{c_3} 3X, X \xrightarrow{c_4} \lambda$, where A, B are two chemicals that are given as input and always kept at a constant amount, X, Y are the intermediate product chemicals that exhibit oscillations, and λ represents the degradation of species X. The initial amounts

of molecules used for generating the target dynamics for parameter estimation are $A = X = 200$, $B = 600$ and $Y = 300$. In this case, we consider two distinct sets of values for the reaction constants $\{c_1, c_2, c_3, c_4\}$ to be estimated using GAs and PSO. The set $\{1, 5 \cdot 10^{-3}, 2.5 \cdot 10^{-5}, 1.5\}$ gives rise to sustained and periodic oscillations in the species X, Y, while the set $\{1, 5 \cdot 10^{-3}, 2.5 \cdot 10^{-4}, 1.5\}$ gives rise to damped oscillations and quickly drives the system dynamics to a steady-state. The corresponding dynamics are shown (with solid and dashed lines) in Figure 6.5 (top right and bottom, respectively).

6.4.2 GAs and PSO settings

The GAs and PSO formulations used in this work evolve individuals of the same shape: n-length vectors of floating point numbers, where n is the number of rate constants to optimise. That is, each allele of an individual corresponds to one parameter that has to be estimated. The experimental settings and the related choices that we have done for GAs and PSO are described below.

Genetic algorithms. The most commonly used GAs formulation evolves fixed length strings over a finite alphabet, while in our case each allele can contain any floating point value from a limited range. This GAs version is often called *real-valued* or *real-coded* GAs (see, e.g., [204] and Section 2.1). Many sophisticated genetic operators for real-coded GAs have recently been defined (like, for instance, Laplace Crossover, Makinen, Periaux and Toivanen Mutation, Non-Uniform Mutation [48]), but in this study we consider classical operators, like the ones originally defined in [204]: *one point* and *average crossover*, *gaussian mutation*, *range mutation* and *reinitialization*.

One point crossover is similar to the standard GAs crossover defined by Holland in [89]: parents are aligned, one crossover point is selected and substrings are inverted to generate offspring. Average crossover returns one offspring that contains at each position the average values of the parents chromosomes. In this work, crossover is executed with a 0.95 rate. In case crossover is not executed, parents are copied in the next population with no modification (reproduction). Otherwise, one operator among one point crossover and average crossover is chosen with uniform probability distribution.

Gaussian mutation perturbs an allele with a number drawn from a normal distribution with mean μ equal to the current value of that allele and $\sigma = 0.05 \cdot \mu$. Range mutation increments or decrements an allele of a prefixed quantity (equal to 5% of its admissible range size in this work). Reinitialization changes the value currently contained in an allele with an uniformly distributed random number in the admissible range. For each individual in the spring, mutation is applied to each allele with probability $1/n$, where n is

its length. If an allele has to be mutated, one operator among gaussian mutation, range mutation or reinitialization is chosen with uniform probability distribution.

The other parameters that we have used are: population size of 100 individuals; maximum number of generations equal to 100; tournament selection of size 5; elitism (i.e. copy of the best individual unchanged in the next population at each generation).

Particle swarm optimization. Among the different versions of PSO present in literature and explained in Section 2.2, we are particularly interested in the version defined in [23], where the co-evolutionary PSO algorithm aimed at optimizing the values of constants C_1 and C_2 first proposed by Miranda and Fonseca in [135], is investigated. Authors of [23] study this co-evolutionary PSO version for a large set of well-known theoretically hand-tailored problems and hint that it may outperform standard PSO for complex problems. For this reason, here we study two versions of the PSO: the canonical PSO formulation (indicated with PSO1 from now on) and the co-evolutionary Miranda and Fonseca model (PSO2 from now on).

The other experimental settings that we have used for both these PSO versions are: swarm size of 20 particles; maximum number of iterations equal to 500 (so doing, GAs and PSO will have executed the same number of fitness evaluations at the end of a run); the inertia weight w has been linearly decremented from 0.9 to 0.2 with gaussian noise, with average equal to the current w value and $\sigma = 0.05$ (as reported in [201]). In addition, in PSO2 we add a gaussian noise to C_1 and C_2 with $\sigma = 0.1$. For both PSO versions, we have considered velocity ranges of half the size of the rate constants ranges, and when a particle reaches a bound of the admissible interval, its velocity is halved and its direction inverted.

6.4.3 Experimental results

Here we discuss the application of our proposed GAs and PSO versions, for the estimation of rate constants of the systems described in Section 6.4.1, for which we have used the following admissible ranges: $c_1 \in [2.5 \cdot 10^{-5}, 2.5 \cdot 10^{-1}], c_2 \in [1 \cdot 10^{-3}, 10], c_3 \in [5 \cdot 10^{-2}, 5 \cdot 10^{2}]$ for the MM system (three further ranges have been tested for this system, obtaining qualitatively similar results to the ones presented here); $c_1 \in [0.1, 10], c_2 \in [5 \cdot 10^{-4}, 5 \cdot 10^{-2}], c_3 \in [2.5 \cdot 10^{-6}, 2.5 \cdot 10^{-4}], c_4 \in [0.15, 15]$ for the Brussellator system with oscillating regime, and $c_1 \in [0.1, 10], c_2 \in [5 \cdot 10^{-4}, 5 \cdot 10^{-2}], c_3 \in [2.5 \cdot 10^{-5}, 2.5 \cdot 10^{-3}], c_4 \in [0.15, 15]$ for the Brussellator system with damped oscillations. Each range has been chosen such that the lower and upper bounds are two (resp. one) orders of magnitude below and above the target value for MM (resp. Brussellator).

For both GAs and PSO, and for each individual in the population, fitness is evaluated by performing a fixed number Z of independent executions of the tau leaping stochastic algorithm, chosen according to the system dynamics. For each system we report the Average Best Fitness (ABF) against fitness evaluations, and the accumulated number of successful runs at each considered value of the fitness evaluations. Given that it is unlike to obtain a fitness equal to zero (see discussion in Section 6.4.4), a run has been considered successful if at least one individual (that we improperly call an optimal solution) has been found with a fitness value smaller than $1.1 \cdot \overline{f}$, where $\overline{f}$ is the best fitness value obtained among any one of the three algorithms and any one of the executed runs (50 in the following examples).

Michaelis-Menten. Figure 6.2 presents the experimental results returned by GAs, PSO1 and PSO2 for the MM system. These results have been obtained with 50 independent runs of GAs, PSO1, PSO2, and $Z = 10$ tau leaping simulations. In Fig. 6.2(a) we report the ABF and in Fig. 6.2(b)

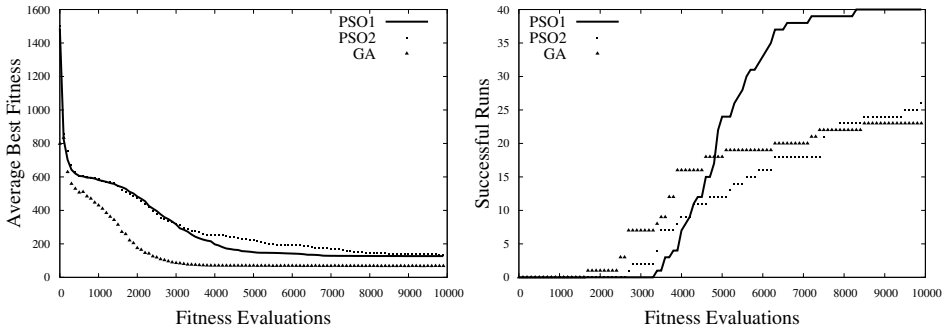


Figure 6.2: Experimental results returned by GAs, PSO1 and PSO2 for the Michaelis-Menten system, obtained by executing 50 independent runs of each algorithm. Left: Average Best Fitness against fitness evaluations. Right: Accumulated number of successful runs against fitness evaluations.

we report the number of successful runs. Figure 6.2(a) shows that GAs have returned a better ABF than the two PSO variants at each value of the fitness evaluations that we have studied, while Fig. 6.2(b) shows that PSO1 has obtained the largest number of successful runs, in particular after 6,000 fitness evaluations. This different behavior between GAs and PSO1 hints that PSO1 has found optimal solutions more frequently than GAs, but when an optimal solution has not been found, GAs have returned individuals of better quality. We also point out that the ABFs of PSO1 and PSO2 are similar to each other (in particular after 9,000 fitness evaluations), while PSO1 has been able to find optimal solutions more often than PSO2.

In Fig. 6.5(a) we report the dynamics of the MM system generated by tau

leaping using the constants found by the best solution generated by PSO1 over the considered 50 runs, which are $c_1 = 0.00245, c_2 = 0.02691, c_3 = 5.03552$. We compare this dynamics (dots) with the target curves (solid lines), pointing out that it very well approximates the targets for each one of the four chemicals involved in the reaction (E, S, ES, P).

Brussellator with oscillating regime. Figure 6.3 reports the experimental results returned by GAs, PSO1 and PSO2 for the Brussellator system with oscillating regime. In particular, Fig. 6.3(a) reports the ABF and

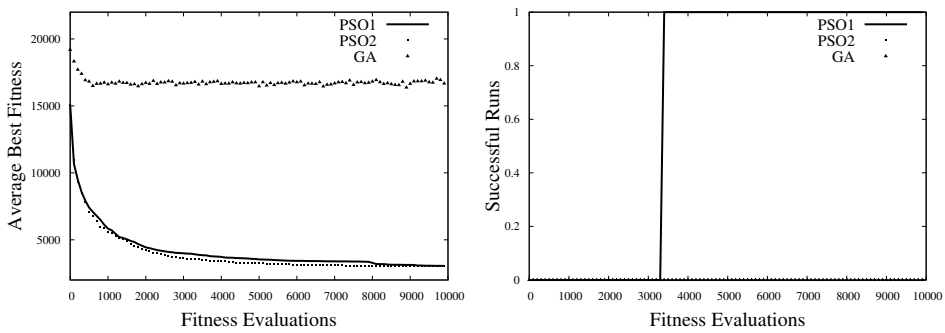


Figure 6.3: Experimental results returned by GAs, PSO1 and PSO2 for the Brussellator system with oscillating regime, obtained by executing 50 independent runs of each algorithm. Left: Average Best Fitness against fitness evaluations. Right: Accumulated number of successful runs against fitness evaluations.

Fig. 6.3(b) reports the number of successful runs. These results have been obtained with 50 independent runs of GAs, PSO1, PSO2; in this case, only one tau leaping execution ($Z = 1$) has been performed because, if we execute more than one run and we calculate their average behaviours, we would risk to flatten the oscillations that characterize this dynamics.

In this case, it is clear that PSO outperforms GAs both from the ABF and success rate viewpoints. In particular, we point out that neither GAs nor PSO2 have been able to find an optimal solution in none of the 50 independent executions that we have performed, while PSO1 has found an optimal solution in only a single run after a number of fitness evaluations approximately equal to 3,500. We conclude that PSO1 can be considered the most suitable algorithm, among the ones that we have studied, for this particular Brussellator dynamics.

Figure 6.5(b) reports the dynamics (dots) over one period of the Brussellator system with oscillating regime that we have obtained using the constants found by the best solution generated by PSO1 over the 50 runs that we have executed. We also observe that the target behavior (lines) has

been reliably approximated with the estimated constants, that are $c_1 = 1.23966, c_2 = 0.00634, c_3 = 4.37171 \cdot 10^{-5}, c_4 = 2.71063$.

Brussellator with damped oscillations. The experimental results returned by GAs, PSO1 and PSO2 for the Brussellator system with steady-state attractor are reported in Fig. 6.4. As for the other cases, in Fig. 6.4(a)

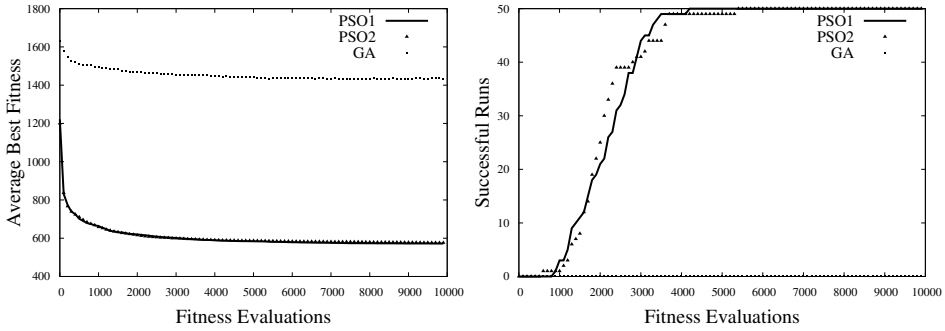


Figure 6.4: Experimental results returned by GAs, PSO1 and PSO2 for the Brussellator system with damped oscillations, obtained by executing 50 independent runs of each algorithm. Left: Average Best Fitness against fitness evaluations. Right: Accumulated number of successful runs against fitness evaluations.

we report the ABF and in Fig. 6.4(b) the accumulated number of successful runs, obtained with 50 independent runs of GAs, PSO1, PSO2, and $Z = 5$ tau leaping simulations.

Also in this case, as for the Brussellator system with oscillating regime, PSO clearly outperforms GAs both for ABF and success rate. In this case, PSO1 and PSO2 show a similar behavior for both these statistics. GAs have not been able to find an optimal solution in none of the 50 independent executions that we have performed, while both PSO variants have found an optimal solution in all the 50 runs after a number of evaluations approximately equal to 5,500.

Given that PSO1 has been able to obtain a success rate equal to 1 with a slightly lower number of fitness evaluations than PSO2, also in this case we report the dynamics obtained using the constants contained in the best solution found by PSO1 ($c_1 = 9.9571, c_2 = 0.00616, c_3 = 0.00033, c_4 = 15$). These dynamics are shown in Fig. 6.5(c). This time the estimated dynamics (dots) of the two chemicals do not approximate the targets dynamics (lines) in a satisfactory way. In particular, we evidence that the estimated dynamics approximates the target at the steady-state, while the first part of the target dynamics, with damped oscillations, is not reliably reconstructed. We hypothesize that this problem could be solved by assigning to the sec-

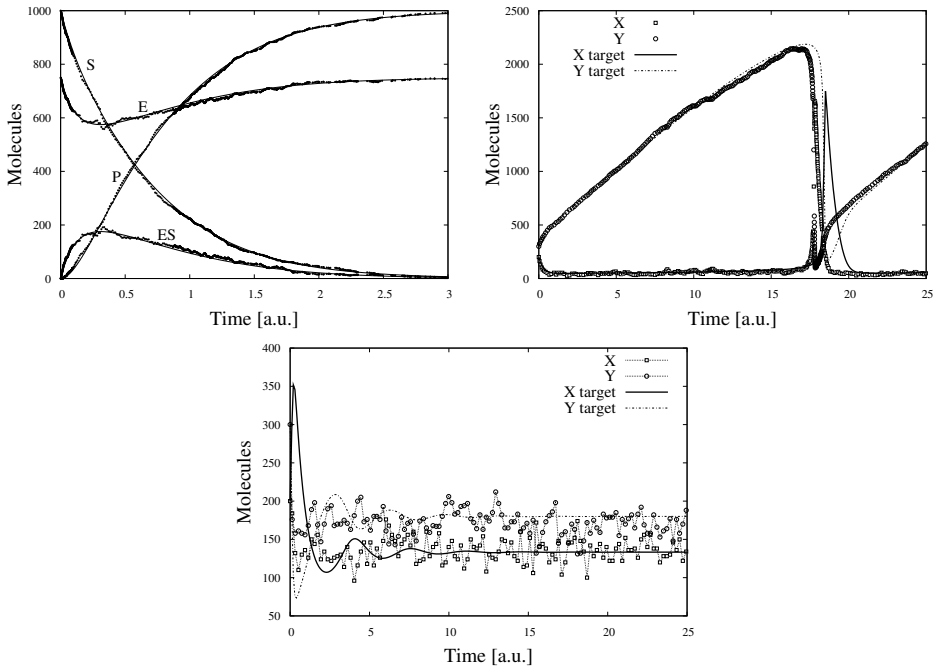


Figure 6.5: Dynamics of the studied systems using the constants found by the best solution generated by PSO1. Left: Michaelis-Menten system; Right: Brussellator with oscillating regime; Bottom: Brussellator with damped oscillations.

ond (non-oscillatory) part of the dynamics a lower weight than to the first (oscillatory) one in the fitness calculation.

6.4.4 Discussion

The problem of estimating the rate constants of two well known biochemical systems has been tackled in this section, using GAs and two different versions of PSO. The experimental results that we have obtained hint that canonical PSO is a suitable optimization method for this problem, since it outperforms both GAs and the PSO version that co-evolves parameters C_1 and C_2. The values of the constants returned by PSO have allowed us to faithfully reconstruct the behavior of the MM system and of the oscillating Brussellator. Nevertheless, for the damped Brussellator even the best set of constants found by PSO has not allowed us to reconstruct the target dynamics in a suitable way. Thus, even though interesting, the presented results deserve further investigations.

Some possible improvements to the proposed optimization methods include the use of further GAs genetic operators, for instance, Laplace Cross-

over, Makinen, Periaux and Toivanen Mutation, Non-Uniform Mutation [48] and the other operators described in Section 2.1, or other PSO models, like PSO with structured populations, with various different neighbourhood structures or with more than two basins of attraction (in addition to the local and global best positions).

Another possible improvement concerns the fitness function. The definition of a suitable fitness function is an intrinsically difficult task for parameter estimation of biochemical systems, for several reasons. First of all, if we exploit stochastic simulation algorithms for the fitness calculation, then it is very unlike to reach the (ideal) value of zero because of the stochastic fluctuations of the estimated time series. Moreover, the same individual will generally have different fitness values each time it is evaluated. These drawbacks could be mitigated by, for instance, executing many times the tau leaping algorithm and calculating averages, but this might be very time consuming in some cases and it could raise serious problems when the target dynamics shows an oscillating behavior (averages may flatten oscillations).

In addition, regarding the fitness function, our choice of using tau leaping to generate the dynamics corresponding to a given set of constants (and thus to evaluate the fitness of an individual) has suggested us to consider the area between the target and the estimated curves, instead of calculating their point-to-point mutual distance. In fact, since the tau leaping algorithm samples points at arbitrary time instants, we have to deal with irregular timing patterns; furthermore, some portions of the dynamics might be more frequently sampled than others. If we use the point-to-point distance to calculate fitness, all sampled points would receive the same importance, thus overrating the dynamics portions that have been more frequently sampled. On the other hand, the area calculation allows us to better evaluate the dynamics also in regions that have not been sampled by tau leaping, and thus approximate more faithfully the (ideal) situation in which we have information for each possible time instant.

To try and solve these problems, this fitness definition could be further improved, for instance, in two ways. The first consists in partitioning the time axis into subintervals, and performing the optimization by incrementally extending the time interval considered for the fitness calculation. The second consists in giving different weights to regions of the dynamics that show distinct behaviours, according to some previous knowledge of the system. Moreover, a different weight could be assigned to the various species involved in the chemical reactions under consideration according, for instance, to the relevance they hold within the system.

Finally, we have recently applied the PSO versions presented here for the parameter estimation of the chemotaxis model described in Section 5.4.1; the preliminary, but promising results of this work can be found in [13].

6.5 Stochastic simulations on a grid framework for parameter sweep applications in biological models

In this section we present the distribution of a large number of computations of τ-DPP stochastic simulator (see [30] and Chapter 4 for additional details), on the EGEE grid framework [56], to enable an analysis of the parameter space of a biochemical system by means of parameter sweep applications (PSAs) [140]. This is done to the aim of exploring this high–dimensional research space, and gain knowledge about the influence of the initial conditions on the system's dynamics.

Given the biological model to analyse, the first step is to define the ranges and the distribution of the parameters values that will be perturbed during the PSA. The most intuitive way to define a set of parametrisations involves the cartesian product of the admissible ranges of each parameter; however, since we have to deal with a high number of parameters, the number of different parametrisation needed to cover the entire research space would be troublesome. A more efficient method to sample this multidimensional space exploits the quasi-random series [149], also called low discrepancy sequences. The discrepancy of a sequence is a measure of its uniformity, and is computed by comparing the actual number of sample points in a given multidimensional space with the number of sample points that "should be" there assuming a uniform distribution. Therefore, the aim of quasi-random series is to "uniformely" cover the space with "few" samples (i.e., with a lower number of points with respect to classic uniform distributions).

Since each instance of a PSA – corresponding to a simulation of the biochemical system by means of τ-DPP –2 is independent, the grid computing framework constitutes a good solution to tackle the high computational cost of this kind of application. The advantage of using a grid distributed approach for computing large computational challenges relies on the high-end scalability of this technology. If jobs are completely independent the theoretical scalability of the system is linear. This is not true in real computations, due to the time needed to schedule jobs, to transfer data and to resubmit failed jobs. However, while distributed platform are not suitable to parallelise the execution of programs that require high connectivity, they are very reliable for data parallel applications, by splitting the computation of input data in independent processes and collecting back the results.

As a case study, we present a PSA for the exploration of the stochastic constants values space of a model composed by 59 reactions which describes the bacterial chemotaxis process presented in Section 5.4.1. This approach represents a benchmark for the development of a sensitivity analysis tool for stochastic biochemical systems implemented on the grid framework.

6.5.1 The EGEE grid platform

In this work we exploit the EGEE project infrastructure, a wide area grid platform for scientific applications, composed of thousands of CPUs, which implements the Virtual Organisation (VO) paradigm [63]. The production framework is a large multi-science grid infrastructure, federating 250 resource centres world-wide, which provides comprehensively 20.000 CPUs and several Petabytes of storage. This infrastructure is used on a daily basis by thousands of scientists federated in over 200 VOs.

The EGEE platform relies on the gLite middleware [103] used in several projects like: DataGrid, DataTag, Globus, GriPhyN, and LCG. The gLite distribution is an integrated set of components designed to enable resource sharing and must be installed on a local server, the User Interface (UI), to allow users to manage the EGEE grid computations. In particular, employing gLite, it is possible to submit jobs to the grid, to monitor their state of advancement, and to retrieve the outputs when the computations have a normal termination or to resubmit the job in case of failure. This grid infrastructure is highly scalable and allows computational intensive challenges to be accomplished, but users must cope with the continuous dynamic reshape of the available resources, which is typical of loosely coupled distributed platforms.

The grid middleware offers a well-established security system, which enables a secure connection to remote resources. This system relies on the Grid Security Infrastructure (GSI) which uses public key cryptography to recognise users. The access to remote clusters is granted by a Personal Certificate, encoded in the X.509 format, associated to each job with the aim to authenticate the user. Moreover, users must be authorised to submit jobs by a VO, that is, a grid community having similar tasks, which grants for them. In this test we join the *Biomed* VO, which shares on average 2000 CPUs and admits applications in the medical image processing, bioinformatics, and more generally the biomedical data processing fields.

The resources available on the EGEE project platform are composed by a network of several Computing Elements (CEs), which are gateways for computer clusters where jobs are actually performed, and an equal number of Storage Elements (SEs), that implement a distributed filesystem to store temporary files. The computational resources are connected to a Resource Broker (RB) that routes each job on a specific CE taking into account the directives of the submitting script, called JDL (Job Description Language). In detail, the Workload Management System is the RB service which schedules jobs by delivering them to the best matching resource, balancing the computational load, via a Condor-G client [24]. Although this brokering is not configurable by users, it provides high performance: bulk submission allows to submit sets of independent jobs in a much more reliable and compound operation, up to a rate of 50Hz for job submission and 0.5Hz for job

dispatching to the CEs.

Each CE submits the incoming job to a batch system queue (PBS or LFS) which hides the farm of Working Nodes. To handle files over the grid, the gLite middleware provides a set of tools to manage data similarly to a distributed filesystem. These tools allow the data to be replicated into different SEs, which can help to reduce the database upload time during the computations. The RB, in fact, is able to redirect the execution of an application to a CE located as near as possible to the files currently used, hence minimising the communication time. Although large files should be always managed by using the SEs, for both input and output, it is possible to use the *SandBox* to load and download small files directly to the CE. The main difference is that files transferred using the *InputSandBox* and the *OutputSandBox* are temporary stored on the RBs, therefore are managed directly by the middleware (but their size should be less than few MB, otherwise RBs will be rapidly stuck) and cannot be reused. On the contrary, files on the SEs have no limitation in size and availability, but have to be handled directly by users.

6.5.2 PSA over the grid

From the computational point of view, the greatest problem of implementing a PSA over the grid is the dynamic behaviour of the available resources. Due to network and system errors, or in relation to the global computational load, the available resources are continuously reshaped and the rate of failure in computations is quite high.

Some solutions, such as Nimrod [22] and APST [28], have been developed to handle these PSAs on grid but they rely on particular middleware implementations, which are not gLite compliant. Moreover, a grid-*inspired* solution to distribute stochastic simulations is described in [108], but the approach does not rely on a grid implementation. While the infrastructure used for the PSAs presented in the following sections is a standard production environment, our focus is not on the middleware implementation, but on testing the reliability of the proposed simulation approach over this distributed platform, by choosing the best strategy to cope with the grid ensemble. Therefore, a crucial point is the creation of a system able to interact with the grid to manage the whole PSA computation, which should check the consistency of each job in a fault tolerant environment.

Managing jobs over the grid: the Challenge Control System. In the context of the EGEE grid, the *Challenge Control System* (CCS) [130] was developed to completely coordinate a computational challenge running on a single UI, by submitting and managing the whole set of jobs in which the computation is split dealing with CEs and SEs. In other words, this framework provides an automatic management of all the necessary opera-

tions to fulfil each single task, because it wraps the grid middleware low level *API* for file handing (*lcg-**) and for job submission, status monitoring and output retrieving (*glite-**), by providing a user-friendly and fault-tolerant environment.

The CCS is highly customisable, thanks to its double layered infrastructure: the first layer was developed to cope with the latest middleware versions of the EGEE infrastructure to manage each single job, while the second level deals with the different requirements of the current application to split the computation and perform the specific tasks on the remote resources. These layers are interconnected by a MySQL database designed to collect all the information needed to manage each grid job, which works as back-end of the system.

The first layer of the CCS works in close connection with the UI and it is mainly devoted to the management of each single job, from the definition of the JDL script and its submission, to the retrieving of the output results. This layer employs the *network time protocol* using the same servers of the EGEE infrastructure, in order to be synchronised with respect to all the grid components, which is crucial for enabling a correct survey on computational trends. Our system also makes use of the *crond* daemon which beats the interval between each round of Challenge Coordination System execution in which, according to the information stored in the MySQL database, tasks such as the submission of new jobs, polling the RB about the status of scheduled jobs, the resubmission of failed jobs and the retrieve back of output results are accomplished.

The second layer is the most important from the application point of view because it coordinates, by using a set of scripts, the job distribution over the grid by splitting the input data in specific tasks: their computation on remote resources and the collection of the output results. Moreover, the second layer checks the output consistence of each simulations computed on the grid platform in connection with the upper layer, which monitors each job as an independent unit.

Implementations to distribute the PSA over the grid. A crucial factor in the implementation of a grid application is the identification of a suitable strategy for splitting the computation into a set of grid jobs, which corresponds to the granularity of the computation. In fact, the computation of long jobs on the grid may cause significant data loss in case of system failure or problems during the data transfer. On the other hand, the execution of a large number of short jobs raises the total latency time in the batch queues, affecting the global performance of the system. Moreover, the size of the output results should be considered, due to the impact of the transfer time on the total computation efficiency. For this reason we performed different PSAs, by varying the number of jobs and the number of

simulations performed in each job, and by altering the computation time by using different strategies of parameters selection, which consequently affect the output size.

In order to enable the execution of a large number of τ-DPP simulations over the grid platform, the CCS [130] has been adapted to satisfy the application requirements. While the lower layer has been periodically updated to be compliant with the latest release of the EGEE grid middleware, the upper layer has been customised for this specific application. In detail, two scripts of the developed framework have been modified to manage the input and output of the PSA computation: one concerns the input management and the other coordinates the computation on the remote resource.

The first script splits of the computation in grid jobs according to the desired granularity. This script populates a directory in which all the files for each single job are temporary collected for submission (i.e., the τ-DPP program and a folder tree containing the input files for each single simulation) and creates the JDL script which describes the job. It then calls the lower layer for the real submission to the grid, specifying some important parameters such as the maximum number of resubmission in case of failure (which allows to overcome most of the problems due to the dynamic reshape of the grid facilities), the maximum time for the job to be queued on a grid cluster before its deletion and resubmission (to avoid over-crowded computational resources), and the output directory on the local server where results will be collected.

The second script of the CCS that has to be customised, is the one effectively executed on the remote clusters. This script manages the input files: it unpacks the input from the *InputSandBox*, defines the pipeline of operations to be accomplished on the remote resources (in this case, it iterates the execution of τ-DPP according to the specified granularity), rebuilds the output directory in a structure which allows an easy evaluation of the results, packs the results to be retrieved and transfers the output results.

The output of a PSA is constituted by a set of calculated dynamics. In this case, the SEs must be used to archive the numerical results before downloading them to the UI. This approach is essential when there is the need of retrieving the complete numerical results for a complete investigation of the system dynamics. However, preliminary analysis of the grid performance suggested that the output data size have a significant effect on the overhead and the success rate of the grid infrastructure.

In order to test the grid infrastructure in a wider range of conditions, we developed another approach in which the analysis of the dynamics is done just after the simulations. In this case, the output is reduced significantly: instead of the complete time series for each molecular species, a value representing an interesting property of the considered dynamics is retrieved immediately through the *OutputSandBox*, without using the SEs. In other words, we developed two different implementations: the first one (imple-

Table 6.1: The 59 reactions of the model of bacterial chemotaxis. The methylation states (m) corresponding to the reactions are: $m = 0$ for reactions $1 - 4$, $m = 0, \ldots, 3$ for reactions $5 - 8$ and $38 - 41$, $m = 1, \ldots, 4$ for reactions $9 - 12$ and $42 - 57$, and $m = 0, \ldots, 4$ for reactions $13 - 37$.

	Reagents	**Products**
1	2MCP^m+2CheW	2MCP^m::2CheW
2	2MCP^m::2CheW	2MCP^m+2CheW
3	2MCP^m::2CheW+2CheA	2MCP^m::2CheW::2CheA
4	2MCP^m::2CheW::2CheA	2MCP^m::2CheW+2CheA
5-8	2MCP^m::2CheW::2CheA+CheR	2MCP^{m+1}::2CheW::2CheA+CheR
9-12	2MCP^m::2CheW::2CheA+CheBp	2MCP^{m-1}::2CheW::2CheA+CheBp
13-17	2MCP^m::2CheW::2CheA+ATP	2MCP^m::2CheW::2CheAp
18-22	2MCP^m::2CheW::2CheAp+CheY	2MCP^m::2CheW::2CheA+CheYp
23-27	2MCP^m::2CheW::2CheAp+CheB	2MCP^m::2CheW::2CheA+CheBp
28-32	lig + 2MCP^m::2CheW::2CheA	lig::2MCP^m::2CheW::2CheA
33-37	lig::2MCP^m::2CheW::2CheA	lig+2MCP^m::2CheW::2CheA
38-41	lig::2MCP^m::2CheW::2CheA+CheR	lig::2MCP^{m+1}::2CheW::2CheA+CheR
42-45	lig::2MCP^m::2CheW::2CheA+CheBp	lig::2MCP^{m-1}::2CheW::2CheA+CheBp
46-49	lig::2MCP^m::2CheW::2CheA+ATP	lig::2MCP^m::2CheW::2CheAp
50-53	lig::2MCP^m::2CheW::2CheAp+CheY	lig::2MCP^m::2CheW::2CheA+CheYp
54-57	lig::2MCP^m::2CheW::2CheAp+CheB	lig::2MCP^m::2CheW::2CheA+CheBp
58	CheYp+CheZ	CheY+CheZ
59	CheBp	CheB

mentation A), in which all the temporal series resulting from a stochastic simulation are retrieved; the second one (implementation B), in which each simulation result is analysed within the corresponding remote resource and only the result of the analysis is retrieved.

6.5.3 Bacterial chemotaxis: a case study

In order to test the performances of PSA over the grid, we have analysed the stochastic constants space of a chemotactic model (see Section 5.4 for the system description, and for additional details about the model and the results obtained by means of stochastic simulations). Note that, actually, the model considered here and used during the PSA is different from that presented in Section 5.4; this is a previous version composed of 59 chemical reactions (with respect to the 62 reactions of the new version), which are listed in Table 6.1.
The reference dynamics (used in the fitness function during the PSA) have been obtained by performing stochastic simulations with the τ-DPP algorithm, and considering the temporal evolution of phosphorylated state of the protein CheY (CheYp). The initial conditions of the system are that reported in Section 5.4, while the set of stochastic constants used for the 59 re-

actions is: $\bar{\mathbf{c}} = (c_1 = 0.1,\ c_2 = 0.01,\ c_3 = 0.1,\ c_4 = 0.02,\ c_5 = 5.0{\cdot}10^{-7},\ c_6 = 5.0 \cdot 10^{-4},\ c_7 = 2.0 \cdot 10^{-4},\ c_8 = 0.0080,\ c_9 = 1.0,\ c_{10} = 0.6,\ c_{11} = 15.0,\ c_{12} = 0.35,\ c_{13} = 5.0 \cdot 10^{-7},\ c_{14} = 5.0 \cdot 10^{-4},\ c_{15} = 2.0 \cdot 10^{-4},\ c_{16} = 6.0 \cdot 10^{-4},\ c_{17} = 0.325,\ c_{18} = 7.0 \cdot 10^{-6},\ c_{19} = 0.0035,\ c_{20} = 0.0014,\ c_{21} = 0.0044,\ c_{22} = 0.8,\ c_{23} = 15.0,\ c_{24} = 0.325,\ c_{25} = 7.0 \cdot 10^{-6},\ c_{26} = 0.0035,\ c_{27} = 0.0014,\ c_{28} = 0.0044,\ c_{29} = 0.29,\ c_{30} = 2.8 \cdot 10^{-5},\ c_{31} = 0.014,\ c_{32} = 0.0056,\ c_{33} = 0.0175,\ c_{34} = 1.0,\ c_{35} = 15.0,\ c_{36} = 0.29,\ c_{37} = 2.8 \cdot 10^{-5},\ c_{38} = 0.014,\ c_{39} = 0.0056,\ c_{40} = 0.0175,\ c_{41} = 0.165,\ c_{42} = 5.0{\cdot}10^{-5},\ c_{43} = 0.025,\ c_{44} = 0.01,\ c_{45} = 0.0306,\ c_{46} = 1.2,\ c_{47} = 15.0,\ c_{48} = 0.165,\ c_{49} = 5.0 \cdot 10^{-5},\ c_{50} = 0.03,\ c_{51} = 0.0112,\ c_{52} = 0.0343,\ c_{53} = 0.05,\ c_{54} = 6.8 \cdot 10^{-5},\ c_{55} = 0.0336,\ c_{56} = 0.0135,\ c_{57} = 0.035,\ c_{58} = 1.4,\ c_{59} = 15.0)$.

6.5.4 Results

For each specific PSA we decided to perform in every single job a constant number of simulations of the system, in order to keep constant the expected CPU time, t_e^{cpu}, for each job within every single PSA. However, the number of simulations and jobs is different in each PSA, according to the purpose of each test and to test the grid using different settings. Therefore, PSAs differ with each other for what concerns the number of simulations grouped in each job and for the number of grid jobs, which have comprehensively a t_e^{cpu} between 0.5 and 3.5 hours. Jobs within this interval are defined *long jobs*, in opposition to *medium jobs* (between 5 and 45 minutes) and *short jobs* (less than 5 minutes). Long jobs are largely considered the most suitable to exploit grid computing because they represent a good trade-off between latency and failure problems [69]. Therefore, the proposed job duration interval is appropriate to value the efficiency of grid in this PSAs challenge.

Four PSAs, in the following called PSA_1, PSA_2, PSA_3 and PSA_4, have been performed on the bacterial chemotaxis model described in Section 6.5.3. The parametrisations of each PSA have been defined by perturbing one or more values of the set $\mathbf{c} = (c_1, \ldots, c_{59})$ of stochastic constants of the model, with respect to the values in $\bar{\mathbf{c}}$ (used to obtain the target dynamics).

The results of the stochastic simulations performed on the grid have been analysed by comparing them to the target dynamics, exploiting the fitness function described in Section 6.3.

PSA_1 and PSA_2. In PSA_1 and PSA_2, only one element of the stochastic constants vector $\mathbf{c}$ has been perturbed in each parametrisation. In PSA_1, for each parameter $c_j \in \mathbf{c}$, 10 simulations have been run using 10 values of c_j linearly distributed within the interval $[0.5 \cdot \bar{c}_j, 1.5 \cdot \bar{c}_j]$, where $\bar{c}_j$ is the reference value of each constant c_j, $j = 1, \ldots, 59$. A total of 590 simulations have been distributed over the grid, organised in jobs of 10 simulations each. Every simulation has been initialised with a relative long time length, 10 time

units, in order to check for potential late effects on the system dynamics. The computation of a single job, corresponding to the set of simulations in which one value c_j has been perturbed 10 times, had t_e^{cpu} of about 45' and was associated to a data volume of about 70MB (leading to a total data volume of 4GB).

The results of PSA_1 denote that there are three most influential parameters: c_8, c_{37}, c_{49} that affect the system dynamics. The variation of the other stochastic constants has negligible effects on the system behaviour, therefore we decided, for the subsequent PSA analysis, to extend the range of variation for all the stochastic constants and to have a finer grain sampling.

PSA_2 was composed of 5900 instances, organised in 59 jobs in which each parameter c_j has been perturbed 100 times, logarithmically distributed within the interval $[10^{-1} \cdot \bar{c}_j, 10 \cdot \bar{c}_j]$. Every job had t_e^{cpu} of about 90' and was associated to a data volume of about 25MB. The lower data volume with respect to PSA_1, despite the higher number of instances, was obtained reducing the total time of the simulated dynamics (conversely, we would have had an output of about 40GB).

The results of PSA_2 confirmed those obtained from PSA_1. As depicted in Figure 6.6, parameters c_8, c_{37}, c_{49} highly influence the system dynamics, while the other stochastic constants have a little effect on the system behaviour. These results show that the crucial points of the model, considering these ranges for the parameters, are represented by reactions r_8, r_{37}, r_{49}. This analysis can be extended increasing the ranges of the parameters variation in order to find the boundaries of the "wild type" behaviour.

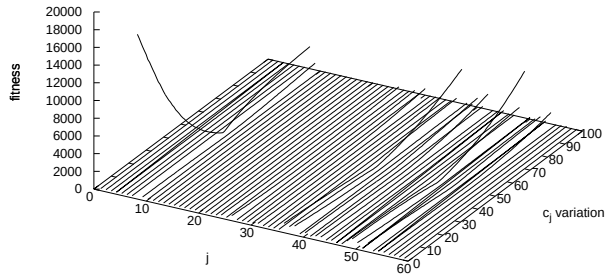


Figure 6.6: Fitness values obtained varying a single parameter for each parametrisation. On the x axis there is the index j of the 59 stochastic constants c_j and on the y axis the relative intensity of the variation with respect to $\bar{c}_j$.

PSA$_3$ and PSA$_4$. PSA$_3$ and PSA$_4$ were composed of 10000 simulations each, the samples have been obtained by using a quasi-random number generator, within the interval $[10^{-1} \cdot \bar{c}_j, 10 \cdot \bar{c}_j]$. 100 jobs, of 100 simulations each, have been distributed on the grid.

In PSA$_3$ every job had t_e^{cpu} of about 230' and produced an output of 188MB in size, while PSA$_4$ was characterised by jobs with t_e^{cpu} of about 30' and produced 12MB in size. The differences in space occupation and computation time are related to the different number of perturbed parameters: 59 in PSA$_3$ and 3 in PSA$_4$. Moreover, these results show that, by (randomly) sampling 59 parameters, a choice of values that lead to faster dynamics, occurs more frequently. The effect of this parameters values (using a stochastic algorithm) consists in a higher number of steps needed to perform a simulation, and hence in higher resources occupation and output size.

The results of PSA$_3$ show that, even though the parametrisation is obtained quasi-randomly sampling all the 59 parameters, the most influent parameter is c_8. This is evident comparing Figures 6.7 and 6.8 – where we show the fitness landscape of the 10000 simulations considering the parameters c_8, c_{37} and c_8, c_{49}, respectively – with Figure 6.9, where the fitness landscape is plotted versus parameters c_{37}, c_{49}. The effect of parameter c_8 induces a sort of ordering of the fitness values. In particular, as the parameter value decreases, the fitness value increases.

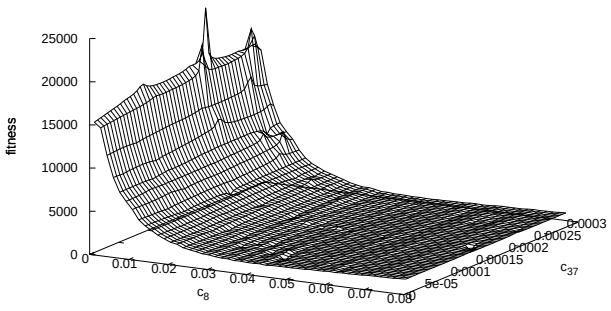


Figure 6.7: Fitness landscape of PSA$_3$ related to parameters c_8 and c_{37}.

In PSA$_4$ we performed 10000 simulations, where the parametrisations occurred varying only the three most influent parameters of the system. The aim of PSA$_4$ was to deeply investigate the 3-dimensional space delimited by parameters c_8, c_{37} and c_{49}. The obtained results show again the influence of parameter c_8 (Figures 6.10 and 6.11 against Figure 6.12).

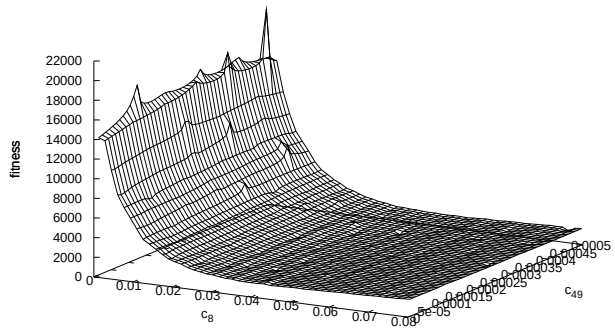


Figure 6.8: Fitness landscape of PSA_3 related to parameters c_8 and c_{49}.

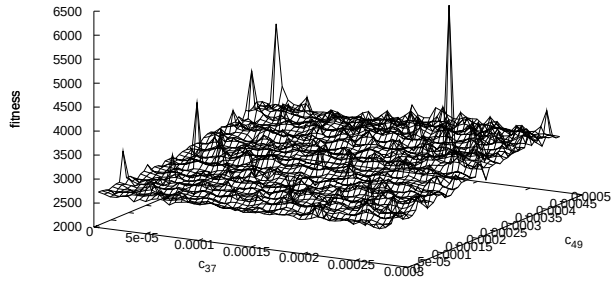


Figure 6.9: Fitness landscape of PSA_3 related to parameters c_{37} and c_{49}.

6.5.5 Performance discussion

In this section we discuss the performance of the PSA executed over the grid by using the two implementations described above.

Implementation A. As stated above, in the implementation A the fitness analysis was performed on the UI after the computation of all the simulations on the grid platform. The four PSAs have comprehensively an estimated computational time of 24 days employing a single CPU while over the grid the full computation lasted 2 days. Hence the crunching factor, C_f, is equal

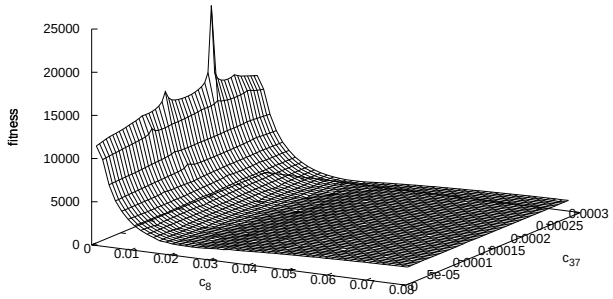


Figure 6.10: Fitness landscape of PSA_4 related to parameters c_8 and c_{37}.

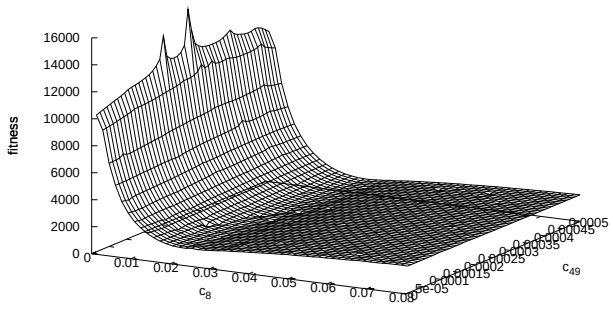


Figure 6.11: Fitness landscape of PSA_4 related to parameters c_8 and c_{49}.

to 12^2.

This result was considerably lower than what we expected. Investigating the problem, we discovered that the number of CPUs used concurrently was significantly higher, but due to unforeseen failure, the scalability was considerably reduced. Considering these four PSAs, a total of 318 jobs have been submitted to the EGEE [56] grid infrastructure: there were about 67% of the jobs reported as successfully finished according to the status logged in the RB, at the first submission. However, the ratio went down

[2] C_f is a metric of the parallelisation gain, defined as the ratio between the total CPU time and the duration of the experiment over grid. It basically represents the average number of CPUs used simultaneously along the computation.

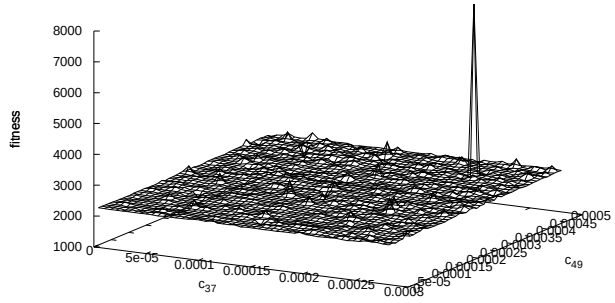


Figure 6.12: Fitness landscape of PSA_4 related to parameters c_{37} and c_{49}.

to 57% after checking the existence of the output. Generally, among the most frequent reported problems there are: (1) faults in the RB scheduling, because resources with the required characteristics are not available; (2) faults in the jobs management by the CE queue, due to overload problems incorrectly reported to the grid; (3) problems with the SE files holding. In our tests, the main cause of job failure seems to be the data transfer between the SEs and the CEs and, in few cases, the unexpected termination of the jobs due to errors on the CEs.

Implementation B. In this implementation the fitness is computed directly on the grid infrastructure, avoiding the full download of the output data describing the system dynamics. Computing the fitness during the simulations, the output was reduced to less than 1 MB, which can be held using the *OutputSandBox*. Implementation B has a success rate of 78%, according to the RB, which outcomes in about 75% of results correctly retrieved. The expected computational time was also of about 24 days on single CPU, because the fitness calculation is quite fast, but the whole computation takes only 30 hours, which correspond to $C_f = 20$.

Comparison between implementations A and B. The comparison of the two implementations allows to better explore the EGEE grid infrastructure potential, by avoiding the bottleneck represented by the fragility of the distributed filesystem, and exploiting in the meantime as much as possible the power of its computational resources. Figure 6.13 shows the effective CPU time on grid resources, t_g^{cpu}, and the total grid time spent to accomplish a single job of a particular PSA, t_g.

It is worthy to note that even if the expected CPU time for jobs within

each PSA is similar, the effective CPU times are distributed on large ranges (for instance, the CPU time for PSA_1 ranges from 10 to 100 minutes) due to the heterogeneity of the computational resources. For what concerns the grid computation time, the graphs in Figure 6.13 show that the grid overhead in implementation A has a large impact on the performance of the system. On the other hand, by preventing the use of the distributed filesystem for the output management, the scalability and the robustness of the system is largely improved (as shown by the shorter grid time of implementation B).

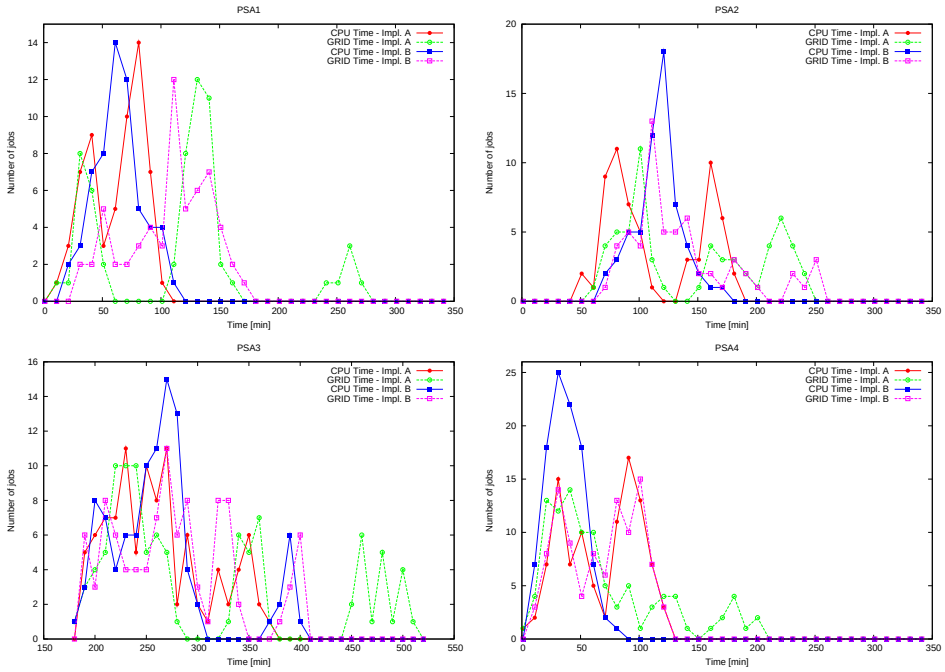


Figure 6.13: Distribution of the jobs of each PSA in relation to their CPU time over grid (t_g^{cpu}) and total grid time (t_g) in implementations A and B.

Considering the job resubmission by the CCS, in the implementation B jobs have been resubmitted at most 3 times before a successful termination, which is considerably better than implementation A, in which some jobs were resubmitted up to 5 times. The other causes of job failure can be hardly eliminated, because they are due to unrecoverable hardware problems (such as hard disk burns, RAM and motherboard failures, or power supply discontinuities), and network access disruption or misconfiguration (note that all these hardware and network problems can lead to *many* job failures). On the other hand, considering that the latency time t_g^l – which includes input file upload, job routing by the RB and the time spent on the grid clusters queue – is quite the same for jobs of the different PSAs (Figure 6.14), the improvement of the grid performance of implementation B with respect to implementation A (shown in Figure 6.13) can be effectively attributed

to the lower failure rate obtained by avoiding the use of the distributed filesystem.

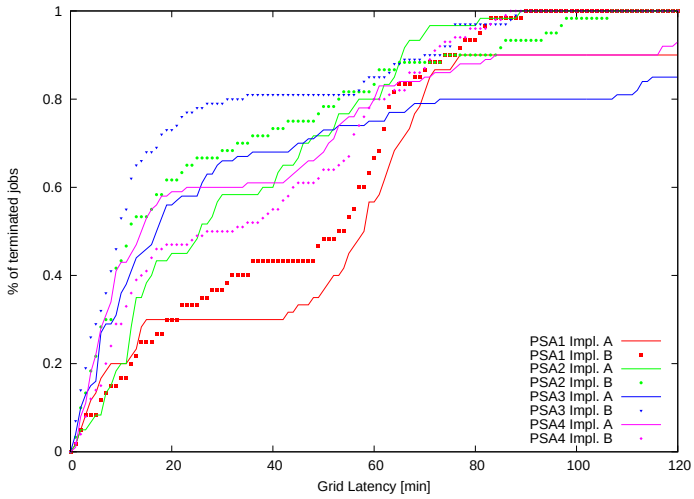


Figure 6.14: Cumulative distribution function of the latency time (t_g^l) for the PSAs. No significant differences are reported in the overall time required for file upload, job routing by the RB and time spent on the grid clusters queue between different implementations.

A very important indicator to evaluate the grid efficiency is the overhead ratio, which can be defined as $O_r = (t_g^l + t_g^{cpu})/t_g^{cpu}$. In some cases the overhead ratio is computed by using t_e^{cpu} (at the denominator), however here we use t_g^{cpu}, thus including the time required to transfer the results from the CEs to the SEs. Moreover, t_g^{cpu} is largely influenced by the grid facility on which the job is performed and can result considerably different from the expected time that we have obtained from our preliminary tests. However, O_r indicates the time spent "on the grid" with respect to the effective t_g^{cpu}. From Figure 6.15 it is clear that the higher the t_g^{cpu} the better O_r. In other words, the use of the grid is justified when the computational time is considerably long, because for short jobs t_g^l can be very large with respect to the effective t_g^{cpu}. Therefore, the empirical idea we have exposed about the importance of the job granularity is confirmed by the obtained results: while there is a considerable failure rate which suggests to perform short tasks, the job duration should be carefully considered with respect to the large overhead of short jobs which prevents from fully exploit the grid potentiality.

The time t_g^{ui} needed for the output retrieval (which is sent to the UI) is shown in Figure 6.16. In the implementation A, t_g^{ui} corresponds to the time required to transfer files from the SEs to the UI, while in the implementation B, t_g^{ui} is the time to retrieve the *OutputSandBox* from the RB. Clearly, the

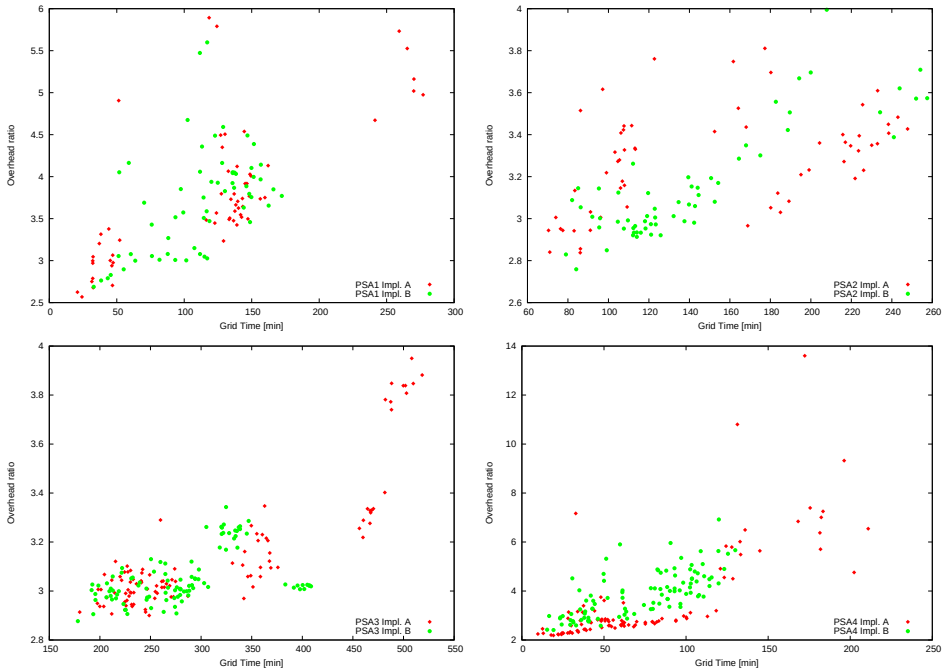


Figure 6.15: Overhead ratio (O_r) for each PSA, best values of the overhead ratio are obtained for large CPU time.

latter is considerably shorter than the time needed to transfer the entire result of the stochastic simulation from the grid distributed filesystem. t_g^{ui} has not been included in the previous performance analysis because it is affected by a noise caused by the intervals between each polling of the RB by the CCS. In particular, in our PSAs the time intervals have been set to 10 minutes in order to avoid an overload of requests to the grid infrastructure.

The rational behind the choice of excluding t_g^{ui} from the performance analysis is that the output is retrieved on the UI only when its status is reported as successfully finished, and a direct interrogation of the RB is needed to be acknowledged of this status. Stated in other words, while t_g^l is calculated as difference between the job submission time and the job execution beginning time, and t_g^{cpu} is calculated as difference between the job execution beginning time and job execution end time, the time elapsed for the effective presence of the output results on the UI (t_g^{ui}) is an uncertain component, because of the interval between two consecutive interrogations of the RB.

The time we can effectively measure is the total time needed to transfer the files from the grid infrastructure to our UI. Due to the authentication on the grid facilities and the interrogation of the distributed filesystem, t_g^{ui} has a large overhead which is considerable high for small files but decreases while dealing with large data. For example, files of few MB have a transfer rate of

about 33KB/sec, files of tens of MB have transfer rate of about 100KB/sec and files larger than 100MB have a transfer rate of about 200KB/sec.

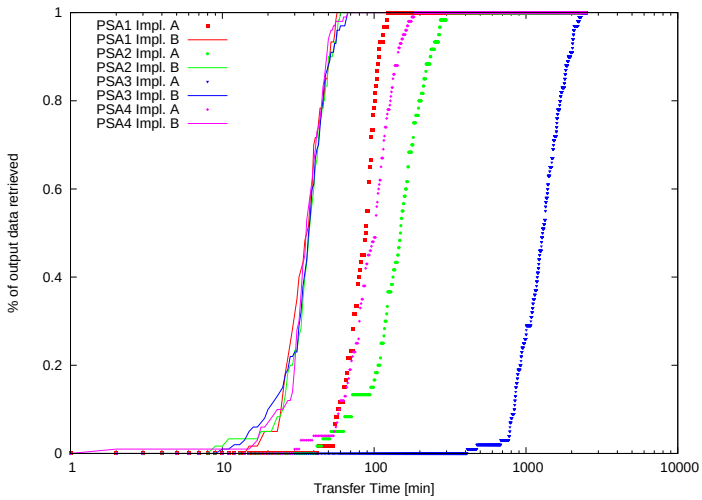


Figure 6.16: Cumulative distribution function of the files transfer time between the grid and the UI (t_g^{ui}) for the different PSAs. In implementation A, t_g^{ui} corresponds to the time needed to transfer the files from the SE to the UI, while in the implementation B it is the time required to retrieve the *OutputSandBox* from the RB.

6.5.6 Conclusion

PSAs are a useful solution to explore the huge state space of a system, which corresponds to all possible combinations of initial molecular numbers and reaction rate constants values. This exploration provides insights about the relationships between the parameters of the chemical reactions and the evolution of the system. In particular, PSA helps in analysing how the choice of a set of values in the parametrisation determines the change of the set of reachable states. This information can be exploited in the application of different techniques such as parameter estimation or sensitivity analysis. Usually, the computation time required to achieve significant results is very high, but thanks to the independence of the parametrisations executed with a PSA, this technique is suitable for a distributed implementation.

In this context we proposed a work with two aims: from one side we want to prove the effectiveness of PSA on grid showing that it is possible to achieve interesting biological results and, on the other side, we tested the EGEE grid infrastructure to show its performance and its bottleneck for this kind of applications, thus providing useful insights for future works.

Considering the chemotaxis model, results of the PSAs showed that the stochastic constants determining the major impact on the system dynamics

are c_8, c_{37} and c_{49}. These constants control, respectively, the addition of the fourth methyl group to the MCP, the unbinding of the ligand from the MCP with 4 methyl groups, and the phosphorylation of CheA protein bound to the MCP complex with MCP in the highest methylated state. The evidence that the three constants are involved in controlling processes in the highly MCP methylated state underlines the role of the MCP methylations for the functioning of the system.

The EGEE grid proved to be a useful solution for the distribution of PSAs concerning the analysis of the biochemical systems dynamics by means of stochastic simulations. The efficiency of this platform has been demonstrated during of our middle-size test and, considering that the more intensive the computation the more scalable the infrastructure, grid computing can be considered a suitable technology to exploit in the context of large scale biological model analysis. However, due to the high rate of failure, a capable submission and monitoring environment should be set-up in order to appropriately manage the volume of data. Moreover, to our experience, the granularity of the submitted jobs and the use of the SE for managing files are element to carefully consider while using the grid infrastructure.

In this work, we compared the grid performances between two implementations: implementation A, in which the entire results of the stochastic simulations are first retrieved from the SEs and then analysed; implementation B, in which the model dynamics are analysed during the simulation phase and only the results of the analysis are retrieved. Implementation A is useful when the complete numerical data of the simulations need to be saved. In implementation B, the SEs have not been used and this determined an increase of the grid performances. As a consequence, these results encourage the development of a tool for the analysis of model dynamics behaviour, with one (or more) parameters perturbed, distributed over the grid within the PSA and avoiding the retrieval of the whole numerical data representing the simulations.

A possible future development of this work consists in the extension of the range of variation for the stochastic constants, in order to identify the boundaries of the "wild type" behaviour of the system, that is, to identify the values of the stochastic constants in which the system's behaviour actually changes and might show other biologically relevant dynamics. Moreover, to test the robustness of the biological model, we plan to apply PSAs with a parametrisation where also the initial molecular quantities are changed. In conclusion, good performances provided by the grid framework encourage to use this architecture to develop a sensitivity analysis tool for stochastic biochemical systems, since also in this field, a large number of simulations need to be performed in order to quantify how much the parameters of the model can affect the system's behaviour, e.g., to help in recognizing the fragilities or the robustness of the system.

Chapter 7

Conclusion and future work

The work presented in this thesis has been motivated by the need to develop novel methods for the stochastic simulation and analysis of discrete models for biochemical systems.

To this aim, we first considered DPPs [162], a variant of P systems [154] that can be used to model chemical, biological and ecological systems. DPPs rely on the definition of a topological structure (i.e., the membrane structure), that is suitable to represent the hierarchical organisation of the compartments composing a biochemical system; moreover, they use rewriting rules to describe the chemical reactions that modify the chemical species involved in the system, which are in turn represented as multisets of objects. In DPPs probabilities are associated to the rules, and these values vary according to the system state. By exploiting these values, it is possible to provide a description of the system's dynamics, that is, DPPs allow to reproduce the stochastic variations of the elements (i.e. chemical species) occurring in the system. However, this description is only qualitative, in the sense that an effective (physical) time streamline cannot be directly associated to the evolution steps of the system.

On the other hand, a quantitative description of a system's dynamics, in which a time length is associated to each evolution step, can be actually provided by using stochastic simulation algorithms. Among others, one of the most efficient procedure of this type is represented by the tau-leaping algorithm [26]. Though it has been shown that tau-leaping can be successfully used to generate the dynamics of biochemical systems in a very accurate way, one of its limitations is related to its applicability only to systems enclosed within a single volume.

To solve these problems, starting from the notions of DPPs and tau-leaping algorithm, we have developed a novel method which combines their suitable features for the description and simulation of biochemical systems, while overcoming their limitations. So doing, we are now able to obtain quantitative and accurate simulations of the dynamics of biochemical sys-

tems composed by many volumes. This novel procedure is called τ-DPP [30]: it consists in a modified tau-leaping algorithm which is enclosed inside each compartment defined in the membrane structure of DPPs.

A second version of this framework, called Sτ-DPP [29], has then been introduced. Sτ-DPP owns all features of τ-DPP and, in addition, it considers the size of both molecules and compartments, to the purpose of building realise more realistic models and obtain more accurate simulation results for biological systems.

The stochastic framework for multivolume systems could be further improved by considering diffusive processes inside the compartments, which is very important in order to characterise the actual localizations of the chemicals involved in the modelled systems. Moreover, since these simulation methods aim at a close description of the biological reality, compartments whose size changes during the evolution should also be taken into consideration, especially when dealing with cellular processes spanning the cell cycle period.

Another biological aspect that is worth considering is the influence of membrane potential over the processes occurring in the system. For instance, in neurons, the open and close states of ion channels are controlled by the existence of a membrane potential and its variations.

Both algorithms, τ-DPP and Sτ-DPP, have been implemented in C language exploiting MPI to handle parallel computations. So doing, the simulation of a model can be executed on parallel architectures, by assigning to each node (or processor) a single compartment, thus reducing the computational load of the whole simulation. The efficiency of τ-DPP and Sτ-DPP can be even improved by using alternative implementations, with the aim of distributing the simulations on parallel architectures like grid, or on parallel devices like GPUs by using CUDA libraries.

Different examples of biological and chemical systems, described through discrete models and simulated using the τ-DPP algorithm, have been presented in this thesis.

We have described the Ras/cAMP/PKA pathway in the yeast *S. cerevisiae*, focusing on the cytoplasmic regulators of the pathway, and providing stochastic simulations of the synthesis of cAMP and of the cytoplasmic quantities of the pivotal complexes. As a future development, we plan to construct an accurate map of other related cellular phenomena, such as the response of this pathway to nutrients and to intracellular acidification, its connections with other pathways co–involved in glucose signalling, and the downhill nuclear expression of target genes. In particular, in order to gain a higher biological relevance of our model, we will include the Gpr1/Gpa2 pathway, which is a specific signalling mechanism that responds only to high glucose concentration, and operates in an addictive redundant way with Ras protein to activate pathway. We believe that establishing the pivotal roles of the different components in the Ras/cAMP/PKA pathway in yeast might

have a positive outcome for the elucidation of similar components in higher eukaryotes. For instance, it is well known that the protein neurofibromin 1 (NF1), acting as tumor suppressor in human cells, is homologous to the proteins of the Ira family, and it has been shown to contain a GAP related domain which regulates Ras [10].

Then, we have proposed a multivolume model implementing a synthetic multicellular clock [67], obtained by coupling the repressilator system with bacterial quorum sensing. This intercellular communication mechanism is able to lead the *local* genetic oscillators (within a noisy and nonidentical population) to *global* oscillatory rhythms. The results of the simulations performed by means of τ-DPP have shown the oscillatory dynamics of the proteins involved in a single cell. Moreover, we have investigated the effects of the quorum sensing signal, focusing on the communication mechanism from the cell to the environment and vice-versa. It would be interesting to extend this work in order to check whether a more complex system, consisting of a population of coupled cells, can show (emergent) synchronisation events with respect to the oscillations of the three repressor proteins and the quorum sensing protein. Since the modelling and simulation approach we have used is stochastic, the gain of such a synchronised behaviour in a noisy system is harder than doing this with a deterministic approach.

The last biological application of τ-DPP proposed in this thesis concerns a very detailed mechanistic model of the bacterial chemotaxis pathway, that takes into account all proteins (and their respective interactions) involved in both signalling and response. In particular, all post-translational modifications of proteins, such as methylation and phosphorylation, have been considered because of their relevant roles in the feedback control mechanisms governing this pathway. We have investigated the dynamics of the pivotal protein involved in chemotaxis, CheYp, under different conditions, such as the deletion of other chemotactic proteins, the addition of distinct amounts of external ligand, the effect of different methylation states. Then, we have investigated the possible influence of stochastic fluctuations of the chemotactic protein CheYp with respect to the running and tumbling motion of bacterial cells, by considering an increasing number of flagella in the individual bacterium. To this aim, we have defined a procedure to identify the synchronisation of rotation of each and every flagella: we have assumed that the cell is sensitive to a threshold level of CheYp, which is evaluated as the mean value of CheYp at steady state. Because of stochastic fluctuations, the amount of CheYp will randomly switch from below to above this value, thus reversing the rotation of each flagellum from counterclockwise to clockwise mode. Our results have demonstrated the importance of stochasticity, showing the link between the synchronisation of all flagella to the stochastic fluctuations of CheYp, as the core component that stands at the basis of chemotactic motions. This model might be improved in order to consider other relevant features of bacterial chemotaxis. For instance, the gradient

of CheYp that can be present inside the cytoplasm – due to the diffusion from the area of its phosphorylation (close to chemotactic receptors) to the area of its activity (around the flagellar motors) – can be a significant aspect in chemotaxis, together with the localization of CheZ (that controls the dephosphorylation of CheYp) and of the flagella around the cell [110].

As a final application, we have considered the framework of chemical computing for the implementation of logic gates and circuits. The chemical computing approaches present in literature define simple logic components as sets of chemical reactions enclosed in a single volume, which can be simulated by means of classic stochastic algorithms. τ-DPP provides an extension of these approaches, since it allows to simulate a set of simple logic components by splitting each of them in separate – but communicating – compartments. So doing, we can both reproduce the behaviour of every component, and propagating the output/input signals by sending/receiving molecules among the distinct compartments.

In particular, we started by simulating two simple logic gates, NAND and XOR. These gates represent the basic elements for the implementation of more complex computing devices. In fact, by composing a number of NAND or XOR logic gates, arranged according to a particular structure, it is possible to represent any boolean function. Then, we have shown how to model and simulate the Fredkin gate and Fredkin circuits. A possible improvement to the implementation and simulation of the presented gates and circuits might include, for instance, the test of different sets of stochastic constants for the chemical reactions, in order to reduce the delay generated during the computation.

A different example of chemical computing that we have presented within the τ-DPP framework, consists in the simulation of the register machine instructions. We have shown the modelling of the SUB instruction as a system composed by two membranes, governed by a conditional behaviour that checks whether a register is zero or not. We have then presented the results obtained by simulating the SUBADD module.

The results obtained from these applications have shown a close correspondence between τ-DPP and the chemical reacting systems occurring inside micro reactors [87]; this can be considered a preliminary step towards a real implementation of τ-DPP by means of the microflow reactors technology.

In the second part of the thesis we have explained the important role of parameters in biochemical systems, by focusing on the set of stochastic constants associated to the reactions. We have described two techniques whose aim are, on the one hand, to calibrate the parameters values by means of optimisation algorithms and, on the other hand, to explore the parameters space by considering a very large set of independent simulations. These methods are called *parameter estimation* and *parameter sweep application*,

respectively.

For the first method, we have compared the performances of genetic algorithms and particle swarm optimizer. The experimental results that we have obtained hint that the canonical implementation of particle swarm optimizer is a suitable optimization method for parameter estimation. The values of the constants returned by particle swarm optimizer have allowed to faithfully reconstruct the behavior of simple chemical systems.

Some possible improvements to the proposed optimisation methods include the use of further operators of genetic algorithms as, for instance, Laplace Crossover, Makinen, Periaux and Toivanen Mutation, Non-Uniform Mutation [48], or other models of particle swarm optimizer, like the implementations with structured populations, with various different neighbourhood structures or with more than two basins of attraction (in addition to the local and global best positions).

Another possible improvement concerns the fitness function. Indeed, the definition of a suitable fitness function is an intrinsically difficult task for parameter estimation of biochemical systems, for several reasons. First of all, if we exploit stochastic simulation algorithms for the fitness calculation, then it is very unlike to reach the (ideal) value of zero because of the stochastic fluctuations of the estimated time series. Moreover, the same individual will generally have different fitness values each time it is evaluated. These drawbacks could be mitigated by executing many times the simulation algorithm and calculating averages, but this might be very time consuming in some cases and could raise serious problems when the target dynamics shows an oscillating behavior (averages may flatten oscillations).

These problems could be solved by introducing a novel version of the fitness function. For instance, the time axis can be divided into subintervals, and the optimization can be performed by incrementally extending the time interval considered for the fitness calculation. Furthermore, different weights can be assigned to the regions of the dynamics that show distinct behaviours, according to some previous knowledge of the system. In addition, a different weight could be assigned to the various species involved in the chemical reactions under consideration, according to the relevance they hold within the system.

Finally, in the last part of the thesis, the parameter sweep application implemented on a grid framework has been presented. As a case study, we have applied this technique to the bacterial chemotaxis model, comparing the results obtained from the application of 4 different configurations of the parameter sweep. The obtained results have shown the suitability of the grid framework for this kind of analysis for biochemical systems, and suggest to implement parameter sweep applications that avoids the retrieval of huge numerical data since the data transfer represents the main cause of job failure.

Parameter sweep application can even be exploited to develop a novel

sensitivity analysis approach for stochastic biochemical models. In mathematical modelling of biological or biochemical systems, sensitivity analysis is used to quantify how much the parameters of the model can affect the system's behaviour, e.g., to help in recognizing the fragilities or the robustness of the system. Though it has been traditionally applied to deterministic continuous models, theories and tools for parametric sensitivity of discrete stochastic systems have recently been defined [82, 166], with the aim of capturing the relevant stochastic effects which can occur in small systems. Taking advantage of these theories, we plan to develop a similar analysis methodology within the framework of τ-DPP. So doing, by exploiting all relevant characteristics of membrane systems (discreteness, multicompartmental structure), integrated with the efficiency of stochastic simulation algorithms, and extended with tools for the estimation and the analysis of system's parameters, the framework presented in this thesis can indeed be considered an useful and successful approach for the stochastic modelling and simulation of biochemical systems.

Bibliography

[1] T. Ahn, P. Wang, L. Watson, Y. Cao, C. Shaffer, and W. Baumann. Stochastic Cell Cycle Modeling for Budding Yeast. In G. Wainer, M. Chinni, P. Roman, H. Rajaei, B. Zeigler, and C. Ribbens, editors, *Proc. of Spring Simulation Multiconference: HPC*, pages 569–574, 2009.

[2] H. R. Akçakaya and P. Sjögren-Gulve. Population viability analysis in conservation planning: an overview. *Ecol. Bull.*, 48:9–21, 2000.

[3] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell*. Garland Science, 4th. edition, 2002.

[4] A. Alhazov, R. Freund, and M. Oswald. Cell/symbol complexity of tissue p systems with symport/antiport rules. *Int. J. Found. Comput. S.*, 17(1):3–25, 2006.

[5] U. Alon, M. G. Surette, N. Barkai, and S. Leibler. Robustness in bacterial chemotaxis. *Nature*, 397(6715):168–171, 1999.

[6] A. Arkin, J. Ross, and H. H. McAdams. Stochastic kinetic analysis of developmental pathway bifurcation in phage λ-infected *Escherichia coli* cells. *Genetics*, 149:1633–1648, 1998.

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

[8] J. Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms table of contents*, pages 101–111. L. Erlbaum Associates Inc. Hillsdale, NJ, USA, 1985.

[9] F. K. Balagaddé, H. Song, J. Ozaki, C. H. Collins, M. Barnet, F. H. Arnold, S. R. Quake, and L. You. A synthetic *Escherichia coli* predator-prey ecosystem. *Mol. Syst. Biol.*, 4:187, 2008.

[10] R. Ballester, D. Marchuk, M. Boquski, R. Letcher, M. Wigler, and F. Collins. The NF1 locus encodes a protein functionally related to mammalian GAP and yeast Ira proteins. *Cell*, 63(4):851–859, 1999.

[11] N. Barkai and S. Leibler. Robustness in simple biochemical networks. *Nature*, 387(6636):913–917, 1997.

[12] M. D. Baroni, E. Martegani, P. Monti, and L. Alberghina. Cell size modulation by CDC25 and RAS2 genes in *Saccharomyces cerevisiae*. *Mol. Cell. Biol.*, 9:2715–2723, 1989.

[13] D. Besozzi, P. Cazzaniga, M. Dugo, G. Mauri, and D. Pescini. Stochastic modelling and parameter estimation of bacterial chemotaxis. Poster at Eighth International Conference on Information Processing in Cells and Tissues (IPCAT), Ascona, Switzerland, April 2009.

[14] D. Besozzi, P. Cazzaniga, M. Dugo, D. Pescini, and G. Mauri. A study on the combined interplay between stochastic fluctuations and the number of flagella in bacterial chemotaxis. *EPTCS*, 6:47–62, 2009.

[15] D. Besozzi, P. Cazzaniga, G. Mauri, D. Pescini, and L. Vanneschi. A comparison of genetic algorithms and particle swarm optimization for parameter estimation in stochastic biochemical systems. In *Proc. of EvoBIO*, number 5483 in Lecture Notes on Computer Science, pages 116–127, 2009.

[16] D. Besozzi, P. Cazzaniga, D. Pescini, and G. Mauri. Seasonal variance in P system models for metapopulations. *Prog. Nat. Sci.*, 17(4):392–400, 2007.

[17] D. Besozzi, P. Cazzaniga, D. Pescini, and G. Mauri. Modelling metapopulations with stochastic membrane systems. *Biosystems*, 91(3):499 – 514, 2008.

[18] D. Besozzi, P. Cazzaniga, D. Pescini, and G. Mauri. *Algorithmic Bioprocesses*, chapter A Multivolume Approach to Stochastic Modelling with Membrane Systems. in press.

[19] L. Bianco, F. Fontana, and V. Manca. P systems with reaction maps. *Int. J. Found. Comput. S.*, 17(1):27–48, 2006.

[20] L. Bianco, D. Pescini, P. Siepmann, N. Krasnogor, F. Romero-Campero, and M. Gheorghe. Towards a p systems pseudomonas quorum sensing model. In H. e. a. E. Hoogeboom, editor, *Proc. of 7th International Workshop on Membrane Computing*, volume 4361 of *LNCS*, 2006.

[21] A. Bren and M. Eisenbach. Changing the direction of flagellar rotation in bacteria by modulating the ratio between the rotational states of the switch protein flim. *J. Mol. Biol.*, 312(4):699–709, 2001.

[22] R. Buyya, D. Abramson, and J. Giddy. Nimrod/g: an architecture for a resource management and scheduling system in a global computational grid. In *Proc. Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region*, volume 1, pages 283–289 vol.1, 2000.

[23] S. Cagnoni, L. Vanneschi, A. Azzini, and A. Tettamanzi. A critical assessment of some variants of particle swarm optimization. In *European Workshop on Bio-inspired algorithms for continuous parameter optimisation, EvoNUM'08*, LNCS, pages 565–574. Springer Verlag, 2008.

[24] S. Campana, D. Rebatto, and A. Sciab. Experience with the glite workload management system in atlas monte carlo production on lcg. *J. Phys. Conf. Ser.*, 119, 2008.

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

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

[27] B. Carter and P. Sudbery. Small-sized mutants of *Saccharomyces cerevisiae*. *Genetics*, 96:561–566, 1980.

[28] H. Casanova, F. Berman, G. Obertelli, and R. Wolski. The apples parameter sweep template: User-level middleware for the grid. In *Proc. ACM/IEEE 2000 Conference Supercomputing*, pages 60–60, 2000.

[29] P. Cazzaniga, G. Mauri, L. Milanesi, E. Mosca, and D. Pescini. A novel variant of tissue p systems for the modelling of biochemical systems. In G. Paun, M. Perez-Jimenez, A. Riscos-Nunez, G. Rozenberg, and A. Salomaa, editors, *Proc. of the 10th International Workshop on Membrane Computing*, 2009.

[30] P. Cazzaniga, D. Pescini, D. Besozzi, and G. Mauri. Tau leaping stochastic simulation method in P systems. In H. J. Hoogeboom, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Proc. of the 7th International Workshop on Membrane Computing*, volume 4361, pages 298–313. LNCS, 2006.

[31] P. Cazzaniga, D. Pescini, D. Besozzi, G. Mauri, S. Colombo, and E. Martegani. Modeling and stochastic simulation of the ras/camp/pka pathway in the yeast saccharomyces cerevisiae evidences a key regulatory function for intracellular guanine nucleotides pools. *J. Biotechnol.*, 133(3):377–385, 2008.

[32] P. Cazzaniga, D. Pescini, F. Romero-Campero, D. Besozzi, and G.Mauri. Stochastic approaches in P systems for simulating biological systems. In M. A. Gutiérrez-Naranjo, G. Păun, A. Riscos-Núñez, and F. J. Romero-Campero, editors, *Proc. of the 4th Brainstorming Week on Membrane Computing*, volume I of *RGNC REPORT 02/2006*, pages 145–164, Sevilla, January 30 - February 3 2006. Fénix Editora.

[33] Cellware. `http://www.cellware.org`.

[34] A. Chatterjee and P. Siarry. Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization. *Comp. Oper. Res.*, 33(3):859–871, 2006.

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

[36] F. Ciocchetta and J. Hillston. Bio-pepa: an extension of the process algebra pepa for biochemical networks. *Electr. Notes Theor. Comput. Sci.*, 194(3):103–117, 2008.

[37] S. Colombo, P. Ma, L. Cauwenberg, J. Winderickx, M. Crauwels, A. Teunissen, D. Nauwelaers, J. H. de Winde, M.-F. Gorwa, D. Colavizza, and J. M. Thevelein. Involvement of distinct G-proteins, Gpa2 and Ras, in glucose- and intracellular acidification-induced cAMP signalling in the yeast *Saccharomyces cerevisiae*. *Embo. J.*, 17:3326–3341, 1998.

[38] S. Colombo, D. R. J. M. Thevelein, J. Winderickx, and E. Martegani. Activation state of the Ras2 protein and glucose-induced signaling in *Saccharomyces cerevisiae*. *J. Biol. Chem.*, 279:46715–46722, 2004.

[39] K. A. Connors. *Chemical kinetics: The study of Reaction Rates in Solution*. VCH, 1990.

[40] M. J. Conroy, Y. Cohen, F. C. James, Y. G. Matsinos, and B. Maurer. Parameter estimation, reliability, and model improvement for spatially explicit models of animal populations. *Ecol. Appl.*, 5(1):17–19, 1995.

[41] Copasi. `http://www.copasi.org/tiki-index.php`.

[42] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1990.

[43] V. Danos and C. Laneve. Formal molecular biology. *Theor. Comput. Sci.*, 325(1):69–110, 2004.

[44] L. Davis. *Handbook of genetic algorithms*. 1991.

[45] P. Dayan, L. Abbott, and L. Abbott. *Theoretical neuroscience: Computational and mathematical modeling of neural systems*. MIT Press, 2001.

[46] K. De Jong. An analysis of the behavior of a class of genetic adaptive systems. *Dissert. Abstr. Inter.*, 36, 1975.

[47] A. De Wit. Spatial patterns and spatiotemporal dynamics in chemical systems. *Adv. Chem. Phys.*, 109:435 – 513, 1999.

[48] K. Deep and M. Thakur. A new crossover operator for real coded genetic algorithms. *Appl. Math. Comput.*, 188(1):895–911, 2007.

[49] P. Dittrich. Chemical computing. In P. F. O. M. J.-P. Banatre, J.-L. Giavitto, editor, *Unconventional Programming Paradigms (UPP 2004)*, volume 3566, pages 19–32. Lecture Notes in Computer Science, 2005.

[50] Dizzy. `http://magnet.systemsbiology.net/software/Dizzy/`.

[51] M. Dugo. Modellazione stocastica della chemiotassi batterica. Bachelor thesis, Università degli Studi di Milano, 2009.

[52] J. B. J. Dunning, D. J. Stewart, B. J. Danielson, B. R. Noon, T. L. Root, R. H. Lamberson, and E. Stevens. Spatially explicit population models: current forms and future uses. *Ecol. Appl.*, 5(1):3–11, 1995.

[53] C. M. Dyer, A. S. Vartanian, H. Zhou, and F. W. Dahlquist. A molecular mechanism of bacterial flagellar motor switching. *J. Mol. Biol.*, 388:71–84, 2009.

[54] R. Eberhart and J. Kennedy. A new optimiser using particle swarm theory. In *Proc. of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43. IEEE service center, 1995.

[55] R. Eberhart and Y. Shi. Particle swarm optimization: developments, applications and resources. In *Proceedings of the 2001 congress on evolutionary computation*, volume 1, pages 81–86. Piscataway, NJ, USA: IEEE, 2001.

[56] Egee project. `http://www.eu-egee.org/`.

[57] J. Elf and M. Ehrenberg. Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases. *IEE P. Syst. Biol.*, 1(2):230–236, 2004.

[58] M. B. Elowitz and S. Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, 2000.

[59] M. B. Elowitz, A. J. Levine, E. D. Siggia, and P. S. Swain. Stochastic gene expression in a single cell. *Science*, 297:1183–1186, 2002.

[60] L. Eshelman, R. Caruana, and J. Schaffer. Biases in the crossover landscape. In *Proceedings of the third international conference on Genetic algorithms table of contents*, pages 10–19. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1989.

[61] L. Eshelman and J. Schaffer. Real-coded genetic algorithms and interval schemata. *FOGA 2*, pages 187–202, 1993.

[62] A. Faisal, L. Selen, and D. Wolpert. Noise in the nervous system. *Nat. Rev. Neurosci.*, 9(4):292–303, 2008.

[63] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15:200–222, 2001.

[64] E. Fredkin and T. Toffoli. Conservative logic. *Int. J. Theor. Phys.*, 21(3):219–253, 1982.

[65] R. Freund, G. Păun, and M. J. Pérez-Jiménez. Tissue p systems with channel states. *Theor. Comput. Sci.*, 330(1):101 – 116, 2005.

[66] P. Frisco. The conformon-P system: a molecular and cell biology-inspired computability model. *Theor. Comput. Sci.*, 312(2-3):295–319, 2004.

[67] J. Garcia-Ojalvo, M. B. Elowitz, and S. H. Strogatz. Modeling a synthetic multicellular clock: repressilators coupled by quorum sensing. *PNAS*, 101:10955–10960, 2004.

[68] C. W. Gardiner. *Handbook of Stochastic Methods: for Physics, Chemistry and the Natural Sciences (Springer Series in Synergetics)*. Springer, 3rd edition, 2004.

[69] C. Germain-Renaud, R. Texier, A. Osorio, and C. Loomis. Grid scheduling for interactive analysis. *Stud. Health Tech. Informat.*, 120:25–33, 2006.

[70] S. Ghaemmaghami, W. Huh, K. Bower, R. W. Howson, A. Belle, N. Dephoure, E. K. O'Shea, and J. S. Weissman. Global analysis of protein expression in yeast. *Nature*, 425:737–741, 2003.

[71] M. Gibson. *Computational methods for stochastic biological systems*. PhD thesis, 2000.

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

[73] D. T. Gillespie. General method for numerically simulating stochastic time evolution of coupled chemical-reactions. *Journ. Comput. Phys.*, 22:403–434, 1976.

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

[75] D. T. Gillespie. *Markov Processes: An Introduction for Physical Scientists*. Academic Press, 1991.

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

[77] D. T. Gillespie and L. R. Petzold. Approximate accelerated stochastic simulation of chemically reacting systems. *Journ. Chem. Phys.*, 115:1716–1733, 2001.

[78] D. T. Gillespie and L. R. Petzold. Improved leap-size selection for accelerated stochastic simulation. *Journ. Chem. Phys.*, 119:8229–8234, 2003.

[79] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[80] D. E. Goldberg. Real-coded genetic algorithms, virtual alphabets, and blocking. *Compl. Syst.*, 5:139–167, 1990.

[81] E. Gross, D. Goldberg, and A. Levitzki. Phosphorylation of the *S. cerevisiae* Cdc25 in response to glucose results in its dissociation from Ras. *Nature*, 360:762–765, 1992.

[82] R. Gunawan, Y. Cao, L. Petzold, and F. J. Doyle. Sensitivity analysis of discrete stochastic systems. *Biophys. J.*, 88:2530–2540, 2005.

[83] C. Guus, E. Boender, and H. Romeijn. *Handbook of global optimization*, chapter Stochastic methods, pages 829–869. Kluwer Academic Publishers, 1995.

[84] S. A. Haney and J. R. Broach. Cdc25p, the guanine nucleotide exchange factor for the Ras proteins of *Saccharomyces cerevisiae*, promotes exchange by stabilizing Ras in a nucleotide-free state. *J. Biol. Chem.*, 269:16541–16548, 1994.

[85] I. Hanski. Metapopulation dynamics. *Nature*, 396:41–49, 1998.

[86] A. Hastings and C. L. Wolin. Within-patch dynamics in a metapopulation. *Ecology*, 70(5):1261–1266, 1989.

[87] S. J. Haswell, R. J. Middleton, B. O'Sullivan, V. Skelton, P. Watts, and P. Styring. The application of microreactors to synthetic chemistry. *Chem. Commun.*, 5:391–398, 2001.

[88] F. Herrera, M. Lozano, and J. Verdegay. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artif. Intell. Rev.*, 12(4):265–319, 1998.

[89] J. H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, Michigan, 1975.

[90] M. Howard and A. D. Rutenberg. Pattern formation inside bacteria: fluctuations due to the low copy number of proteins. *Phys. Rev. Lett.*, 90(12):128102, 2003.

[91] M. Inda, A. Belloum, M. Roos, D. Vasunin, C. de Laat, L. Hertzberger, and T. Breit. Interactive Workflows in a Virtual Laboratory for e-Bioscience: The SigWin-Detector Tool for Gene Expression Analysis. In *Second IEEE International Conference on e-Science and Grid Computing, 2006. e-Science'06*, pages 19–19, 2006.

[92] M. Ionescu, G. Păun, and T. Yokomori. Spiking neural P systems. *Fund. Inform.*, 71(2):279–308, 2006.

[93] V. A. A. Jansen. The dynamics of two diffusively coupled predator-prey populations. *Theor. Popul. Biol.*, 59:119–131, 2001.

[94] V. A. A. Jansen and A. L. Lloyd. Local stability analysis of spatially homogeneous solutions of multi-patch systems. *J. Math. Biol.*, 41:232–252, 2000.

[95] S. Janson and M. Middendorf. A hierarchical particle swarm optimizer for dynamic optimization problems. In *Applications of Evolutionary Computing*, pages 513–524. Springer, 2004.

[96] P. Jorgensen, J. Nishikawa, B. Breitkreutz, and M. Tyers. Systematic identification of pathways that couple cell growth and division in yeast. *Science*, 297:395–400, 2002.

[97] M. S. Jurica and B. L. Stoddard. Mind your b's and r's: bacterial chemotaxis, signal transduction and protein recognition. *Structure*, 6(7):809–813, 1998.

[98] J. Kennedy. The behavior of particles. In *Proceedings of the 7th International Conference on Evolutionary Programming VII*, pages 581–589. Springer-Verlag London, UK, 1998.

[99] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proc. of the IEEE International Conference on Neural Networks*, volume IV, pages 1942–1948, Piscataway, NJ, 1995.

[100] B. Kernighan and D. Ritchie. *The C programming language*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1978.

[101] H. Kitano. *Foundations of Systems Biology*, chapter Systems Biology: Toward System-level Understanding of Biological Systems. MIT Press, Cambridge/MA, 2001.

[102] I. Korec. Small universal register machines. *Theor. Comput. Sci.*, 168(2):267–301, 1996.

[103] E. Laure, S. Fisher, A. Frohner, C. Grandi, P. Kunszt, A. Krenek, O. Mulmo, F. Pacini, F. Prelz, J. White, M. Barroso, P. Bunic, F. Hemmer, A. D. Meglio, and A. Edlund. Programming the grid with glite. *Comp. Meth. Sci. Tech.*, 12(1):33–45, 2006.

[104] C. Lenzen, R. Cool, H. Prinz, J. Kuhlmann, and A. Wittinghofer. Kinetic analysis by fluorescence of the interaction between ras and the catalytic domain of the guanine nucleotide exchange factor Cdc25Mm. *Biochemistry*, 37:7420–7430, 1998.

[105] A. Leporati, C. Zandron, and G. Mauri. Reversible p systems to simulate fredkin circuits. *Fund. Inform.*, 74(4):529–548, 2006.

[106] M. D. Levin, C. J. Morton-Firth, W. N. Abouhamad, R. B. Bourret, and D. Bray. Origins of individual swimming behavior in bacteria. *Biophys. J.*, 74(1):175–181, 1998.

[107] R. Levins. Some demographic and genetic consequences of environmental heterogeneity for biological control. *Bull. Ent. Soc. Amer.*, 71:237–240, 1969.

[108] S. Leye, J. Himmelspach, M. Jeschke, R. Ewald, and A. Uhrmacher. A grid-inspired mechanism for coarse-grained experiment execution. In *Proc. 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications DS-RT 2008*, pages 7–16, 2008.

[109] G. Li and R. M. Weis. Covalent modification regulates ligand binding to receptor complexes in the chemosensory system of escherichia coli. *Cell*, 100(3):357–365, 2000.

[110] K. Lipkow, S. S. Andrews, and D. Bray. Simulated diffusion of phosphorylated chey through the cytoplasm of escherichia coli. *J. Bacteriol.*, 187(1):45–53, 2005.

[111] D. Longo and J. Hasty. Dynamics of single-cell gene expression. *Mol. Syst. Biol.*, 2(64):1–10, 2006.

[112] P. Ma, S. Wera, P. V. Dijck, and J. M. Thevelein. The PDE1-encoded low-affinity phosphodiesterase in the yeast *Saccharomyces cerevisiae* has a specific function in controlling agonist-induced cAMP signaling. *Mol. Biol. Cell*, 10:91–104, 1999.

[113] K. Man, K. Tang, and S. Kwong. Genetic algorithms: concepts and applications. *IEEE T. Ind. Electron.*, 43(5):519–534, 1996.

[114] V. Manca. The metabolic algorithm for P systems: principles and applications. *Theor. Comput. Sci.*, 404(1-2):142–155, 2008.

[115] D. Maravall and J. de Lope. Multi-objective dynamic optimization with genetic algorithms for automatic parking. *Soft Comput.*, 11(3):249–257, 2006.

[116] O. Marion. Independent agents in a globalized world modelled by tissue p systems. *Artif. Life Rob.*, 11(2):171–174, 2007.

[117] T. T. Marquez-Lago and K. Burrage. Binomial tau-leap spatial stochastic simulation algorithm for applications in chemical kinetics. *J. Chem. Phys.*, 127(10):104101, 2007.

[118] E. Martegani, P. Cazzaniga, D. Pescini, D. Besozzi, S. Colombo, and G. Mauri. Stochastic modeling of the Ras/cAMP signal transduction pathway in yeast. In *Computational Methods in Systems Biology*, Trento, Italy, October 18-19 2006.

[119] E. Martegani, R. Tisi, F. Belotti, S. Colombo, C. Paiardi, J. Winderickx, P. Cazzaniga, D. Besozzi, and G. Mauri. Identification of an intracellular signalling complex for RAS/cAMP pathway in yeast: experimental evidences and modelling. In *International Specialised Symposium on Yeasts*, Hanasaari - Espoo, Finland, June 18-21 2006. Poster.

[120] C. Martín-Vide, G. Păun, J. Pazos, and A. Rodríguez-Patón. Tissue p systems. *Theor. Comput. Sci.*, 296(2):295–326, 2003.

[121] C. Martín-Vide, J. Pazos, G. Păun, and A. Rodríguez-Patón. A new class of symbolic abstract neural nets: Tissue p systems. In *COCOON '02: Proc. of the 8th Annual International Conference on Computing and Combinatorics*, pages 290–299, London, UK, 2002. Springer-Verlag.

[122] N. Matsumaru, F. Centler, P. S. di Fenizio, and P. Dittrich. Chemical organization theory as a theoretical base for chemical computing. *Int. J. Unconv. Comput.*, 3:285–309, 2005.

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

[124] B. A. Mello and Y. Tu. Perfect and near-perfect adaptation in a model of bacterial chemotaxis. *Biophys. J.*, 84(5):2943–2956, 2003.

[125] P. Mendes. *Foundations of Systems Biology*, chapter Modeling large biochemical systems from functional genomic data: Parameter estimation, pages 163–168. MIT Press, Cambridge/MA, 2001.

[126] P. Mendes and D. B. Kell. On the analysis of the inverse problem of metabolic pathways using artificial neural networks. *Biosystems*, 38(1):15–28, 1996.

[127] P. Mendes and D. B. Kell. Non-linear optimization of biochemical pathways: applications to metabolic engineering and parameter estimation. *Bioinformatics*, 14:869–883, 1998.

[128] T. C. Meng, S. Somani, and P. Dhar. Modeling and simulation of biological systems with stochasticity. *In Silico Biol.*, 4:0024, 2004.

[129] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs*. Springer-Verlag London, UK, 1996.

[130] L. Milanesi, I. Merelli, G. T. P. Cozzi, and A. Orro. *Handbook of Research on Computational Grid Technologies for Life Sciences, Biomedicine and Healthcare*, chapter Functional genomics applications in grid. Medical Information Science Reference, 2009.

[131] B. L. Miller, B. L. Miller, D. E. Goldberg, and D. E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Syst.*, 9:193–212, 1995.

[132] M. B. Miller and B. L. Bassler. Quorum sensing in bacteria. *Annu. Rev. Microbiol.*, 55:165–199, 2001.

[133] R. Milner. *Communicating and mobile systems: the π-calculus*. Cambridge Univ Pr, 1999.

[134] M. L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

[135] V. Miranda and N. Fonseca. Epso best-of-two-worlds meta-heuristic applied to power system problems. In *Proc. IEEE Congress on Evolutionary Computation 2002*, pages 1080–1085. IEEE Computer Society Press, 2002.

[136] A. Moilanen. SPOMSIM: software for stochastic patch occupancy models of metapopulation dynamics. *Ecol. Modell.*, 179:533–550, 2004.

[137] C. G. Moles, P. Mendes, and J. R. Banga. Parameter estimation in biochemical pathways: A comparison of global optimization methods. *Genome Res.*, 13(11):2467–2474, 2003.

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

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

[140] E. Mosca, P. Cazzaniga, I. Merelli, D. Pescini, L. Milanesi, and G. Mauri. Stochastic simulations on a grid framework for parameter sweep applications in biological models. Accepted.

[141] E. Mosca, P. Cazzaniga, D. Pescini, G. Mauri, and L. Milanesi. A stochastic, discrete model for the simulation of the action potential. In *International Specialised Symposium on Yeasts*, Bologna - Italy, November 24–25 2008. Poster.

[142] The MPI standard Web Page. `http://www-unix.mcs.anl.gov/mpi/`.

[143] H. Muhlenbein and D. Schlierkamp-Voosen. Predictive models for the breeder genetic algorithm I. Continuous parameter optimization. *Evol. Comput.*, 1(1):25–49, 1993.

[144] D. Müller, S. Exler, L. Aguilera-Vázquez, E. Guerrero-Martín, and M. Reuss. Cyclic AMP mediates the cell cycle dynamics of energy metabolism in *Saccharomyces cerevisiae*. *Yeast*, 20:351–367, 2003.

[145] J. D. Murray. *Mathematical Biology. I: An introduction*. Springer-Verlag, New York, 2002.

[146] D. Nelson and M. Cox. *Lehninger Principles of Biochemistry*. W. H. Freeman Company, 2004.

[147] J. R. S. Newman, S. Ghaemmaghami, J. Ihmels, D. K. Breslow, M. Noble, J. L. DeRisi, and J. S. Weissman. Single-cell proteomic analysis of *S. cerevisiae* reveals the architecture of biological noise. *Nature*, 441:840–846, 2006.

[148] R. Nicolescu, M. Dinneen, and Y. Kim. Structured modeling with hyperdag p systems. In *Proc. of the 7th Brainstorming week on Membrane Computing*, volume II, pages 85–108, 2009.

[149] H. Niederreiter. *Random number generation and quasi-Monte Carlo methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

[150] J. Nikawa, S. Cameron, T. Toda, K. M. Ferguson, and M. Wigler. Rigorous feedback control of cAMP levels in *Saccharomyces cerevisiae*. *Gene. Dev.*, 1:931–937, 1987.

[151] S. Olabarriaga, A. Nederveen, and B. O'Nuallain. Parameter sweeps for functional mri research in the" virtual laboratory for e-science" project. In *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, pages 685–690. IEEE Computer Society Washington, DC, USA, 2007.

[152] M. R. Parsek, D. L. Val, B. L. Hanzelka, J. E. Cronan, and E. P. Greenberg. Acyl homoserine-lactone quorum-sensing signal generation. *PNAS*, 96(8):4360–4365, 1999.

[153] G. Păun. Computing with membranes. *J. Comput. Syst. Sci.*, 61(1):108–143, 2000. (See also *Turku Center for Computer Science-TUCS Report* No 208, 1998, www.tucs.fi).

[154] G. Păun. *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.

[155] G. Păun and R. Păun. Membrane computing as a framework for modeling economic processes. In *Proc. of the Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 11–18. IEEE Computer Society, 2005.

[156] G. Păun, G. Rozenberg, and A. Salomaa. Membrane computing with external output. *Fund. Inform.*, 41(3):259–266, 2000. (See also *Turku Center for Computer Science-TUCS Report* No 218, 1998, www.tucs.fi).

[157] G. Păun, Y. Sakakibara, and T. Yokomori. P systems on graphs of restricted forms. *Publ. Math-Debrecen*, 60:635–660, 2002.

[158] A. Penttilä, E. Zubko, K. Lumme, K. Muinonen, M. Yurkin, B. Draine, J. Rahola, A. Hoekstra, and Y. Shkuratov. Comparison between discrete dipole implementations and exact techniques. *J. Quant. Spectrosc. Radiat. Transfer*, 106(1-3):417–436, 2007.

[159] M. J. Pérez-Jiménez and F. J. Romero-Campero. *Transactions on Computational Systems Biology VI*, volume 4220/2006 of *Lecture Notes in Computer Science*, chapter P Systems, a New Computational Modelling Tool for Systems Biology, pages 176–197. Springer Berlin / Heidelberg, 2006.

[160] D. Pescini. *Modelling, analysis and stochastic simulations of biological systems*. PhD thesis, Università degli studi di Milano - Bicocca, 2008.

[161] D. Pescini, D. Besozzi, and G. Mauri. Investigating local evolutions in dynamical probabilistic P systems. In *Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'05)*, pages 440–447. IEEE Computer Press, 2005.

[162] D. Pescini, D. Besozzi, G. Mauri, and C. Zandron. Dynamical probabilistic P systems. *Int. J. Found. Comput. S.*, 17(1):183–204, 2006.

[163] D. Pescini, D. Besozzi, C. Zandron, and G. Mauri. Analysis and simulation of dynamics in probabilistic P systems. In N. P. A. Carbone, editor, *Proc. of 11th International Workshop on DNA Computing, DNA11*, volume 3892, pages 236–247, London, ON, Canada, June 6-9, 2005 2006. LNCS.

[164] D. Pescini, P. Cazzaniga, C. Ferretti, and G. Mauri. First steps toward a wet implementation for τ-dpp. In D. Corne, P. Frisco, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Proc. of the 9th International Workshop on Membrane Computing*, volume 5391 of *LNCS*, pages 355–373, 2008.

[165] C. Petri. *Kommunikation mit automata*. PhD thesis, Ph. D. thesis, University of Bonn, Bonn, Germany.(in German), 1962.

[166] S. Plyasunov and A. P. Arkin. Efficient stochastic sensitivity analysis of discrete event systems. *J. Comp. Phys.*, 221:724–738, 2007.

[167] C. W. Pouton, K. M. Wagstaff, D. M. Roth, G. W. Moseley, and D. A. Jans. Targeted delivery to the nucleus. *Adv. Drug. Deliver. Rev.*, 59(8):698–717, 2007.

[168] The p systems web page. `http://psystems.disco.unimib.it/`.

[169] N. Radcliffe. Equivalence class analysis of genetic algorithms. *Complex Syst.*, 5(2):183–205, 1991.

[170] C. V. Rao, J. R. Kirby, and A. P. Arkin. Design and diversity in bacterial chemotaxis: a comparative study in escherichia coli and bacillus subtilis. *PLoS Biol.*, 2(2):E49, 2004.

[171] J. M. Reed and S. H. Levine. A model for behavioral regulation of metapopulation dynamics. *Ecol. Modell.*, 183:411–423, 2005.

[172] C. Reeves. Using genetic algorithms with small populations. In *Proceedings of the 5th International Conference on Genetic Algorithms table of contents*, pages 92–99. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1993.

[173] S. Reinker, R. M. Altman, and J. Timmer. Parameter estimation in stochastic biochemical reactions. *IEE P. Syst. Biol.*, 153:168–178, 2006.

[174] C. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34. ACM New York, NY, USA, 1987.

[175] F. Rolland, J. de Winde, K. Lemaire, E. Boles, J. Thevelein, and W. J. Glucose-induced cAMP signalling in yeast requires both a G-protein coupled receptor system for extracellular glucose detection and a separable hexose kinase-dependent sensing process. *Mol. Microbiol.*, 38:348–358, 2000.

[176] N. Rosenfeld, J. W. Young, U. Alon, P. S. Swain, and M. B. Elowitz. Gene Regulation at the Single-Cell Level. *Science*, 307:1962–1965, 2005.

[177] S. Rudoni, S. Colombo, P. Coccetti, and E. Martegani. Role of guanine nucleotides in the regulation of the Ras/cAMP pathway in *Saccharomyces cerevisiae*. *Biochim. Biophys. Acta*, 1538:181–189, 2001.

[178] T. Runarsson and X. Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Trans. Evol. Comput.*, 4(3):284–294, 2000.

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

[180] G. M. Santangelo. Glucose signaling in *Saccharomyces cerevisiae*. *Microbiol. Mol. Bio. Rev.*, 70(1):253–282, 2006.

[181] B. E. Scharf, K. A. Fahrner, L. Turner, and H. C. Berg. Control of direction of flagellar rotation in bacterial chemotaxis. *PNAS*, 95(1):201–206, 1998.

[182] L. Shapiro and R. Losick. Dynamic spatial regulation in the bacterial cell. *Cell*, 100(1):89–98, 2000.

[183] Y. Shi and R. Eberhart. Parameter selection in particle swarm optimization. *LNCS*, pages 591–600, 1998.

[184] T. S. Shimizu and D. Bray. Modelling the bacterial chemotaxis receptor complex. *Novartis Found. Symp.*, 247:162–77; discussion 177–81, 198–206, 244–52, 2002.

[185] V. Sourjik. Receptor clustering and signal processing in e. coli chemotaxis. *Trends Microbiol.*, 12(12):569–576, 2004.

[186] P. A. Spiro, J. S. Parkinson, and H. G. Othmer. A model of excitation and adaptation in bacterial chemotaxis. *PNAS*, 94(14):7263–7268, 1997.

[187] V. Stankovski and W. Dubitzky. Special section: Data mining in grid computing environments. *Future Gener. Comp. Sy.*, 23(1):31–33, 2007.

[188] Stochkit. `http://engineering.ucsb.edu/~cse/StochKit/`.

[189] A. Stundzia and C. Lumsden. Stochastic simulation of coupled reaction-diffusion processes. *Journ. Comput. Phys.*, 127(1):196–207, 1996.

[190] G. Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms table of contents*, pages 2–9. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1989.

[191] A. D. Taylor. Metapopulations, dispersal, and predator-prey dynamics: an overview. *Ecology*, 71(2):429–433, 1990.

[192] M. Thattai and A. van Oudenaarden. Intrinsic noise in gene regulatory networks. *PNAS*, 98(15):8614, 2001.

[193] J. M. Thevelein. Signal transduction in yeast. *Yeast*, 10:1753–1790, 1994.

[194] J. M. Thevelein, L. Cauwenberg, S. Colombo, J. H. de Winde, M. Donaton, F. Dumortier, L. Kraakman, K. Lemaire, P. Ma, D. Nauwelaers, F. Rolland, A. Teunissen, P. V. Dijck, M. Versele, S. Wera, and J. Winderickx. Nutrient-induced signal transduction through the protein kinase A pathway and its role in the control of metabolism, stress resistance, and growth in yeast. *Enzyme Microb. Technol.*, 26:819–825, 2000.

[195] J. M. Thevelein and J. H. de Winde. Novel sensing mechanisms and targets for the cAMP-protein kinase A pathway in the yeast *Saccharomyces cerevisiae*. *Mol. Microbiol.*, 33:904–918, 1999.

[196] T. Tian and K. Burrage. Binomial leap methods for simulating stochastic chemical kinetics. *Journ. Chem. Phys.*, 121:10356–10364, 2004.

[197] T. Toda, S. Cameron, P. Sass, M. Zoller, and M. Wigler. Three different genes in *S. cerevisiae* encode the catalytic subunits of the cAMP-dependent protein kinase. *Cell*, 50:277–287, 1987.

[198] T. E. Turner, S. Schnell, and K. Burrage. Stochastic approaches for modelling in vivo reactions. *Comput. Biol. Chem.*, 28:165–178, 2004.

[199] J. J. Tyson. Some further studies of nonlinear oscillations in chemical systems. *Journ. Chem. Phys.*, 58:3919–3930, 1973.

[200] R. Unger and J. Moult. Genetic algorithms for protein folding simulations. *J. Mol. Biol.*, 231(1):75–81, 1993.

[201] Y. D. Valle, G. Venayagamoorthy, S. Mohagheghi, J. Hernandez, and R. Harley. Particle swarm optimization: Basic concepts, variants and applications in power systems. *IEEE Trans. Evol. Comput.*, 12(2):171–195, 2008.

[202] G. H. Wadhams and J. P. Armitage. Making sense of it all: bacterial chemotaxis. *Nat. Rev. Mol. Cell Biol.*, 5(12):1024–1037, 2004.

[203] W. W. Weisser, V. A. A. Jansen, and M. P. Hassell. The effects of a pool of dispersers on host-parasitoid systems. *J. Theor. Biol.*, 189:413–425, 1997.

[204] A. H. Wright. Genetic algorithms for real parameter optimization. In G. J. Rawlins, editor, *Foundations of genetic algorithms*, pages 205–218. Morgan Kaufmann, San Mateo, CA, 1991.

[205] Yeast gfp fusion localization database. `http://yeastgfp.ucsf.edu/`.