Evolutionary Inference of Biological Systems Accelerated on Graphics Processing Units

Marco S. Nobile

Ph.D. Program in Computer Science — XXVII Cycle

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano–Bicocca

Advisor: Prof. Giancarlo Mauri

October 2014

Abstract

In silico analysis of biological systems represents a valuable alternative and complementary approach to experimental research. Computational methodologies, indeed, allow to mimic some conditions of cellular processes that might be difficult to dissect by exploiting traditional laboratory techniques, therefore potentially achieving a thorough comprehension of the molecular mechanisms that rule the functioning of cells and organisms. In spite of the benefits that it can bring about in biology, the computational approach still has two main limitations: first, there is often a lack of adequate knowledge on the biological system of interest, which prevents the creation of a proper mathematical model able to produce faithful and quantitative predictions; second, the analysis of the model can require a massive number of simulations and calculations, which are computationally burdensome. The goal of the present thesis is to develop novel computational methodologies to efficiently tackle these two issues, at multiple scales of biological complexity (from single molecular structures to networks of biochemical reactions). The inference of the missing data — related to the three-dimensional structures of proteins, the number and type of chemical species and their mutual interactions, the kinetic parameters — is performed by means of novel methods based on Evolutionary Computation and Swarm Intelligence techniques. General purpose GPU computing has been adopted to reduce the computational time, achieving a relevant speedup with respect to the sequential execution of the same algorithms. The results presented in this thesis show that these novel evolutionary-based and GPU-accelerated methodologies are indeed feasible and advantageous from both the points of view of inference quality and computational performances.

Acknowledgements

Tons of “thank you” go to Viola, for her love, support, and unwavering patience during the last three (four?) years. I would like to express deep gratitude to Professor Giancarlo Mauri, who has always been a caring and supportive supervisor. A huge thank to Daniela Besozzi, for plenty of reasons: for her friendship, for the privilege of her severe mentorship, for shaping me as a scientist, for changing me forever. Many thanks go to Paolo Cazzaniga, for his strength, for the constant help and support, for being the best co-worker I have ever had. Of course, two big thanks go to both Professors Hitoshi Iba and Thomas LaBean, for allowing me to spend amazing days in their research groups at the Tōkyō University and North Carolina State University. Many thanks to all the wonderful colleagues who helped me in those days: Alexandria, Annica, Hasegawa-sensei, Jacob, Nicole, Peng, Shonosuke-san, and, last but not least, Yuki-san (I owe you so much). I also want to send a huge “thank you” to Chie-san, JB, Kei-san, Lydia, Oliver, and Rafaela, who turned my days in Tōkyō into an unforgettable experience. Many thanks to the people at DISCo, University of Milano-Bicocca, who shared with me ideas, beers and nice talks during these 1000 days spent together. It is a pleasure to take this occasion to thank all my co-authors and the scientists I have met and been in contact with, especially at Vanderbilt University. I also want to thank Professors Gabriella Pasi, Fabio Stella, Domenico Sorrenti and Giuseppe Vizzari for their support, helpful suggestions and, in some occasions, that glimpse of additional enthusiasm. Thanks to Davide and Simone for their friendly and precious support during the last months. Thanks to my parents, for always supporting me and for not freaking out when I decided to quit my job to pursue this crazy idea. Thanks to Mara and Lorenzo, too, for their constant help and comprehension. Pixellated thanks go to the 8-bit weirdos of Lack of Bits, Gabriele and Michele: those nights spent playing together definitely helped me through this all. A huge thank to all my close friends, for always being there, and to my beloved Alan and Melody.

A final thank to John Carpenter, for composing the outstanding soundtracks which provided me with the energies to complete this final effort of my doctorate.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

In Silico Biology (ISB) pursues the investigation of biological systems by means of mathematical modeling and computational analysis methods, providing an alternative approach for a thorough comprehension of living organisms, that could be unfeasible by means of classic laboratory experiments only [24, 59]. ISB represents a conceptual container for a variety of disciplines, having different goals and working on biological systems of different granularity. The subjects of this thesis belong to three specific areas: *Systems Biology* (SYSB), *Synthetic Biology* (SB) and *Computational Structural Biology* (CSB).

SYSB mainly focuses on the investigation of cellular systems, trying to characterize their emergent properties, considering a holistic point of view based on the interactions among their fundamental components [8]. One of the foremost goals of SYSB is to provide a computational way to understand the subtle mechanisms of biochemical pathways, which are difficult to dissect by means of experimental research. The main concept behind SYSB is the definition of rigorous mathematical representations of complex dynamical non-linear systems, described at a desired level of detail, laying the foundations for the development of simulators and computational analysis methods. Once that a model has been validated through *ad hoc* laboratory experiments, that is, its simulated dynamics is reliable and characterized by predictive power, it can be used to investigate and analyze the behavior of the system in different conditions, which might be hard to analyze with laboratory experiments. This allows to *understand* its functioning and its response to environmental or structural perturbations [144], paving the way to novel methodologies for the *control* of the system, with possible breakthroughs in bio-medical or bio-engineering applications. In addition, SYSB can aid to formulate new hypotheses, leading to novel laboratory experiments and to further research directions [162], in an iterative cycle of model refinements [59].

SB exploits an approach similar to SYSB to the final goal of designing and constructing novel, modified and improved biological circuits or synthetic organisms, which are able to perform a specific task [23]. The idea of a circuit-like connectivity between biological parts was postulated for the first time nearly 50 years ago [217]. The proof of concept that a computing-like behavior could be applied to biological systems was the design of the first synthetic gene networks, realized by using engineering-based methodologies in the first years of the XXI century [84, 94]. Following this strategy, in many cases SB works by purposely modifying the genome of existing organisms [127], to obtain a new dynamic behavior from an altered gene regulatory network (GRN).

The goal of CSB is the determination and characterization of the structural aspects of biomolecules by means of computational methodologies [179]. This approach paves the way to the development of novel methods, able to exploit structural information to determine the affinity between molecules, ultimately leading to the comprehension of the functional relationships of proteins [240]. This knowledge can be exploited, for instance, to establish their involvement in biological processes, making them attractive targets for drug design [329].

The research activity in these three strongly interdisciplinary fields — SYSB, SB and CSB — requires the synergistic effort of multiple areas including biology, bioinformatics, biotechnology, chemistry, mathematics, medicine, physics, probability, statistics and, of course, computer science. The outcome of these joint efforts is a model of the system under investigation, defined at the proper level of detail. For instance, a network of interacting biomolecules in the case of SYSB; a GRN to be engineered in the case of SB; a protein described as a collection of atoms and their spatial positions in the case of CSB. In this context, Figure 1.1 provides an overview of the topics discussed in the present thesis (graphically represented as hexagons) and their conceptual interconnections. Specifically, light blue hexagons represent the main biological disciplines involved in this work: the ISB macro-area, SYSB, SB, and CSB. Four methodologies are here presented to create, extend, simulate and analyze mathematical models in all these areas.

For the specific case of SYSB and SB, this thesis relies on the conceptual framework of mechanistic modeling, whose aim is to give a detailed description of the molecular mechanisms that rule the interactions between the components of biological systems [24, 50]. Mechanistic models are assumed to represent the most likely candidates to achieve a detailed comprehension of biological systems, since they can lead to quantitative predictions of cellular dynamics. In particular, in this thesis, all the tools developed in the context of SYSB and SB are based on reaction-based models (RBMs),

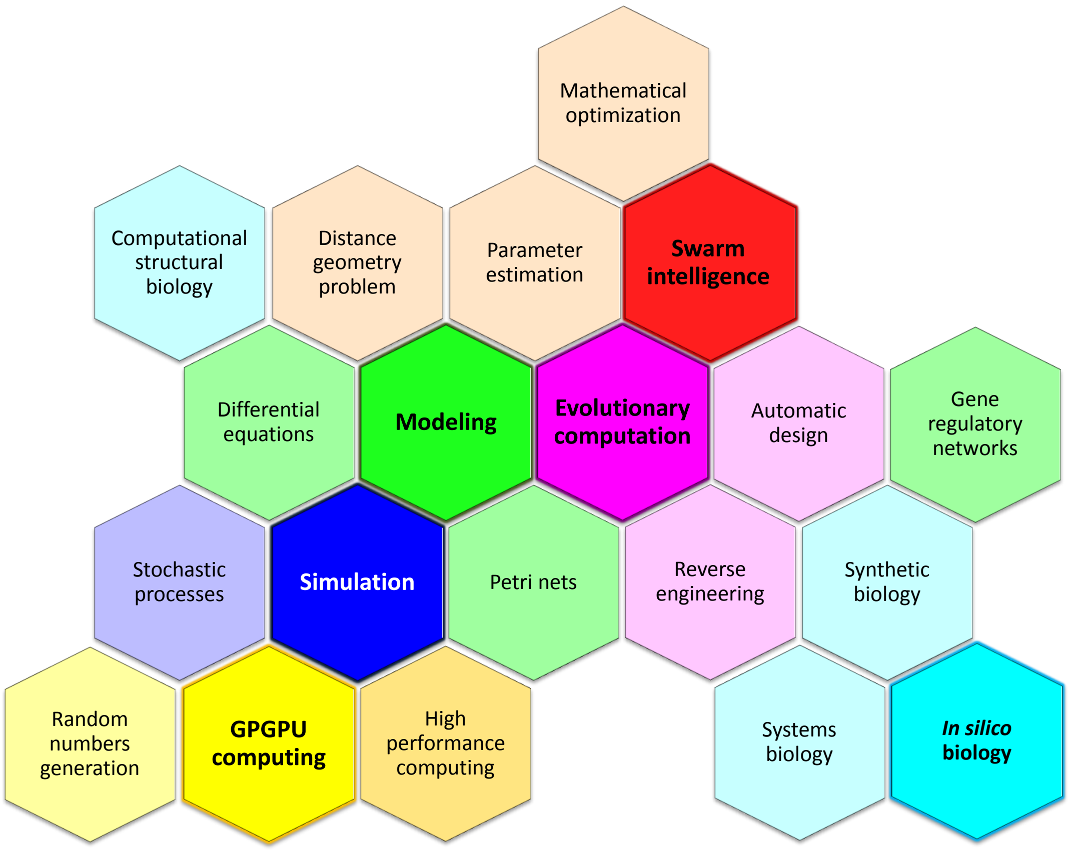


Figure 1.1: Overview of the topics discussed in this thesis and their conceptual interconnections. Different colors represent the main methodologies that are exploited: green for the modeling approaches of biochemical systems (Chapter 2) and blue for their simulation methods (Chapter 5); purple for Evolutionary Computation and Red for Swarm Intelligence (Chapter 3), which are the basis for the estimation and optimization methods; yellow for general-purpose GPU computing (Chapter 4), the high-performance parallel architecture used to accelerate the proposed methodologies. This thesis presents methods for parameter estimation of stochastic systems (Chapter 6), reverse engineering of biochemical pathways (Chapter 7), the automatic design of gene regulatory networks (Chapter 8) and the inference of the tertiary structure of proteins according to partial Nuclear Magnetic Resonance data (Chapter 9). The goal of this work is to provide fast and reliable computational tools to achieve the aforementioned tasks, by integrating all these disciplines in the context of *in silico* biological investigation (light blue).

one of the possible formalisms used to define mechanistic models. The reasons behind this design choice are numerous:

- RBMs are immediately readable and comprehensible. This is particularly relevant when the modeling and inference tools are used by experimental biologists, who are usually not familiar with alternative and more complex mathematical formalisms, like differential equations;

- RBMs can be easily modified or extended during further model refinements (e.g., after the experimental validation), in order to modify either the set of chemical species or the set of reactions defined in the initial formalization;

- the same RBM can be exploited to carry out both stochastic and deterministic simulations of the system dynamics, by exploiting the definition of propensity functions, or the conversion of the set of reactions into the corresponding system of differential equations, according to the mass-action kinetics [59, 101].

In order to perform such simulations and analyses, though, the mechanistic model must be entirely and properly specified. Namely, the model needs to contain all the information about the chemical reactions, the kinetic parameters, the chemical species and the initial conditions (e.g., molecular amounts) of the system. All these information should be as precise as possible, since biochemical models can be characterized by a high sensitivity to the change of even a single kinetic parameter, or the initial state of the system. Thus, high quality kinetic parameters are mandatory to produce quantitative simulations of biochemical systems, turning models into powerful predictive tools.

Unfortunately, kinetic parameters are often missing in literature because it is generally difficult, expensive or even impossible to measure them directly, a circumstance that leads to the definition of the Parameter Estimation problem (PE) [211]. A naïve, yet very diffused, approach to tackle this problem is to manually tune the parameters in the model [344]: an error-prone and time-consuming procedure. An alternative approach consists in exploiting some automatic methodology, which identifies a model parameterization able to reproduce the system dynamics. This problem is challenging, especially when some chemical species are present in low amounts, a condition that leads to the emergence of stochasticity and noisy behavior. A further challenge of the PE problem consists in a proper use of data that are typically measured during laboratory experiments. In biochemical assays, for instance, a small set of sparsely sampled discrete time-series of the chemical species involved in the system is derived, usually executed in different initial conditions (e.g., different nutrients, temperature, etc.) and replicated a certain number of times.

The PE problem assumes that the model of the biochemical process is known, which is not always the case for biological systems that are still lacking an in-depth characterization at the molecular level [16, 74]. When the domain knowledge is insufficient to build a complete model of the biochemical system of interest, a Reverse Engineering (RE) methodology can be employed to infer the missing reactions. Simple interaction networks can be inferred by means of correlation-based methods [16, 43] in which some perturbations of the system (e.g., variation of the concentrations) can be used to calculate correlation coefficients and entropy-based mutual information values, which are exploited to build a putative interaction network. Then, the network is simplified and pruned by identifying and removing indirect interactions. Unfortunately, even though these methodologies can be applied to large-scale systems, interaction networks have limited applicability in SYSB since they do not allow quantitative simulations. As a matter of fact, the only information provided by these networks concerns the chemical species involved in the system and their mutual interactions, without any information about stoichiometry or kinetic parameters. Thus, the RE of mechanistic models is a complex task which, to some extent, "contains" the PE problem.

Seen from a different perspective, the goal of SB is complementary to the RE problem: instead of reverse engineering the behavior of an observable biochemical system, it deals with the development of a novel synthetic system characterized by an arbitrarily chosen dynamics [239]. Differently from RE, though, the synthetic system, able to perform a specific task, is the target of the design and does not exist in nature yet. For this reason, the problem is further complicated by the implicit lack of knowledge about the chemical species (i.e., species not observable in laboratory, hence not belonging to the potential set of targets species, like the intermediate molecular complexes) that are necessary to build a network able to produce the expected dynamics. This thesis describes an automatic method — able to create models of GRNs performing a specific task — not requiring any assumption about the number and type of these unknown chemical species, which represents an absolute novelty in the field.

The inference of the tertiary structure of proteins (i.e., their three-dimensional shape) is a particularly complex topic. These structures can be inferred with multiple approaches, exploiting a variety of input data [73]. In this thesis, this problem is faced by relying on the available information about inter-atomic distances only. This is called the Molecular Distance Geometry Problem (MDGP) [65]. MDGP becomes particularly challenging in the case of incomplete inter-atomic distances, which is typical of data obtained by means of Nuclear Magnetic Resonance (NMR) experiments [360]. In this case, the problem is NP-hard[1] as demonstrated by Saxe *et al.* [299] by reducing a 1-dimensional version of the MDGP to the SUBSETSUM problem [219].

A common trait of all the aforementioned problems is that the inference of the missing knowledge can be reduced to an optimization problem. In such strategy, a

[1]That is, at least as hard as problems solvable in polynomial time on a non-deterministic Turing machine.

search space of the feasible solutions is defined according to the characteristics and the constraints of the problem. The search space is then algorithmically explored, looking for the optimal solution which best fits the target data and minimizes the “difference” between the simulation outcomes and the experimental data. Unfortunately, the search spaces of biological problems are generally huge, multi-modal, noisy and non-linear [59], so that traditional optimization techniques like the Simplex Method or Gradient Descent cannot be employed.

In general, all the optimization problems discussed in this thesis are known to have a relevant difficulty [211, 299], potentially belonging to the NP-hard class. To tackle such complexity, the optimization strategies in this thesis rely on a set of *bio-inspired meta-heuristics*[2]. Specifically, the methods employed in this work are inspired by Darwinian evolution (i.e., Evolutionary Computation [69], EC) and by the emergent intelligence of groups of organisms (i.e., Swarm Intelligence [158], SI).

Both approaches rely on the possibility of quantifying the “goodness” of each candidate solution of the optimization problem, by means of a *fitness function*. In the case of EC methods, a *population* of candidate solutions adapts as a consequence of an evolutionary process, which exploits mechanisms for selection, mutation and crossover. By iteratively applying these genetic operators, the population converges to an optimal solution with respect to the defined fitness function. Differently from EC, SI relies on the intelligence emerging from the collective effort of simple agents, able to explore the search space and share the information inside the group, thus providing an effective means for the optimization.

EC and SI techniques have different characteristics and are suitable for specific problems. Specifically, some evolutionary techniques like Genetic Algorithms and Genetic Programming are generally used to optimize problems whose solutions can be encoded with discrete representations, while SI techniques like Bee Colony Optimization or Particle Swarm Optimization are generally used for real-valued problems. Some problems, though, can be represented by means of a combination of these elements. For instance, BRMs are composed of chemical reactions with discrete stoichiometric coefficients and real-valued kinetic parameters; protein structures can be defined as discrete vectors of atoms, all characterized by real-valued (relative) positions. Even though evolutionary algorithms can be extended to perform the co-evolution of discrete and real-valued elements (e.g., by using the so-called *ephemeral constants* [289]), in this work it is shown that EC and SI can be combined, in order to create hybrid

[2]Meta-heuristics are methodologies for the solution of a computational problem (e.g., optimization) which make no assumptions about it, so that they can be applied to any problem.

"super-algorithms" tackling multiple problems at once and taking the best of these two worlds.

Both EC and SI approaches rely on a randomly created population of candidate-solutions, exploring the search space driven by the fitness function. Since this function is generally based on the simulation outcome of the candidate solution, these approaches are computationally burdensome. Nevertheless, all the fitness evaluations are independent, so that a parallel architecture can be employed to reduce the running time. Among the possible alternatives, all the methods presented in this thesis were designed to exploit the general-purpose GPU (GPGPU) computing. The GPGPU computing is a novel paradigm which provides a low-cost, energy-efficient means to access tera-scale performances on common workstations (and peta-scale performances[3] on GPU-equipped supercomputers [72]), obtained by leveraging the powerful parallel capabilities of modern video cards.

In this thesis, GPU-powered biochemical simulators (both deterministic and stochastic) were designed to accelerate the computational methodologies developed for PE, RE and design of synthetic circuits. These simulators represent valuable tools for the investigation of biochemical systems, and they are vital for the feasibility of the optimization algorithms, since they strongly reduce the computational effort due to the fitness evaluations. As a matter of fact, the reduction of the simulation time allowed by the GPU acceleration permits a deeper investigation of cellular systems, not only for the solution of PE, RE and ED problems. Indeed, to gain novel insights into the functioning of a biological system under physiological or perturbed conditions, the application of many computational methods requires the execution of a large number of simulations. Examples of time consuming tasks, relying on a massive number of simulations, are parameter sweep analysis [235] or sensitivity analysis [49]. Thanks to the GPU implementations presented in this thesis, researchers can perform this large amount of simulations on a local workstation, without the need for traditionally large, expensive and dedicate high-performance computing infrastructures (e.g., CPUs clusters, GRID).

*

* *

[3]Tera- and peta-scale refer to computing systems whose peak performances reaches one teraflop and petaflop, respectively. They correspond to 10^{12} and 10^{15} floating point operations per second.

This Ph.D. thesis consists of two parts: Part I, which provides the necessary theoretical background; Part II, which introduces the novel work developed during the three years of my doctorate program.

Part I starts with **Chapter 2**, where the rationale behind the use of mechanistic modeling as the foundation for the whole work is explained in details. **Chapter 3** provides a summary of the most widespread optimization methodologies, focusing on EC and SI techniques. **Chapter 4** introduces the GPGPU computing and, specifically, the CUDA architecture exploited throughout the whole work.

Part II begins with **Chapter 5**, which introduces three GPU-powered biochemical simulators exploited by the inference methodologies. Specifically, Section 5.1 deals with the development of a deterministic biochemical parallel simulator based on mass-action kinetics named cupSODA; Section 5.2 presents an extended version of the previous simulator, which allows to consider a different type of kinetics (namely, Hill kinetics), to massively simulate a model of the blood coagulation cascade; finally, Section 5.3 describes the development of cuTauLeaping, a stochastic simulator of biochemical systems based on the tau-leaping algorithm. cuTauLeaping represents the basis for cuPEPSO, a PE methodology based on SI, described in **Chapter 6**. **Chapter 7** introduces cuRE, a PE methodology relying on cupSODA and combining EC and SI. An automatic methodology for the design of synthetic biological circuits, named cuGENED, is discussed in **Chapter 8**. **Chapter 9** introduces MemHPG, a hybrid memetic algorithm combining EC and SI, able to tackle the MDGP with incomplete information. As a conclusion of the work, **Chapter 10** discusses the strenghts, drawbacks and potential improvements of the methodologies presented in this thesis, while **Chapter 11** concludes with a final overview and future developments.

This thesis also contains three Appendices. **Appendix A** presents different RBMs of biochemical systems, used throughout the whole thesis to test the proposed methodologies. **Appendix B** reports a comprehensive list of all the acronyms used in the thesis, while **Appendix C** provides the reader with a complete list of the symbols used in the formalizations, with a brief explanation of their semantics.

Scientific production

Journal papers

- Nobile M.S., Cazzaniga P., Besozzi D., Cipolla D., Mauri G.
  Parameter Estimation on Graphics Processing Units: a Multi-swarm Approach for Stochastic Cellular Systems
  Submitted to IEEE Transactions on Evolutionary Computation, 2014

- Cazzaniga P., Damiani C., Besozzi D., Colombo R., Nobile M.S., Gaglio D., Pescini D., Molinari S., Mauri G., Alberghina L., Vanoni M.
  Computational Strategies for a System-level Understanding of Metabolism
  Accepted to Metabolites, 2014 (under revision)

- Nobile M.S., Cazzaniga P., Besozzi D., Pescini D., Mauri G.
  cuTauLeaping: a GPU-powered Tau-leaping Stochastic Simulator for Massive Parallel Analyses of Biological Systems
  PLOS ONE, Volume 9, Issue 3: e91963, 2014

- Cazzaniga P., Nobile M.S., Besozzi D., Bellini M., Mauri G.
  Massive Exploration of Perturbed Conditions of the Blood Coagulation Cascade through GPU Parallelization
  BioMed Research International, Volume 2014: Article ID 863298, 2014

- Nobile M.S., Cazzaniga P., Besozzi D., Mauri G.
  GPU-accelerated Simulations of Mass-action Kinetics Models with cupSODA
  The Journal of Supercomputing, Volume 69, Issue 1: pp. 17–24, 2014

- Bellini M., Besozzi D., Cazzaniga P., Mauri G., Nobile M.S.
  Modeling and Simulation and Analysis of the Blood Coagulation Cascade Accelerated on GPU
  Egyptian Computer Science Journal, Volume 37, Issue 7: pp. 10–23, 2014

Conference proceedings

- Nobile M.S., Citrolo A.G., Cazzaniga P., Besozzi D., Mauri G.
  A Memetic Hybrid Method for the Molecular Distance Geometry Problem with Incomplete Information

Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC2014), Beijing (China), published by IEEE, pp. 1014—1021, 2014

- Nobile M.S.
Evolutionary Inference of Biochemical Interaction Networks Accelerated on Graphics Processing Units
Proceedings of the 11th International Conference on High Performance Computing & Simulation 2013 (HPCS 2013), Helsinki (Finland), published by IEEE, pp. 668–670, 2013

- Nobile M.S., Besozzi D., Cazzaniga P., Mauri G.
The Foundation of Evolutionary Petri Nets
Proceedings of the 4th International Workshop on Biological Processes & Petri Nets (BioPPN 2013), a satellite event of PETRI NETS 2013 (G. Balbo and M. Heiner, eds.), CEUR Workshop Proceedings, Volume 988, pp. 60–74, 2013

- Nobile M.S., Besozzi D., Cazzaniga P., Mauri G., Pescini D.
Reverse Engineering of Kinetic Reaction Networks by Means of Cartesian Genetic Programming and Particle Swarm Optimization
Proceedings of the 2013 IEEE Congress on Evolutionary Computation (CEC2013), Cancun (Mexico), published by IEEE, Volume 1, pp. 1594–1601, 2013

- Nobile M.S., Besozzi D., Cazzaniga P., Mauri G., Pescini D.
cupSODA: a CUDA-Powered Simulator of Mass-action Kinetics
Proceedings of the 12th International Conference on Parallel Computing Technologies (PaCT), Saint Petersburg (Russia). V. Malyshkin (Ed.). Lecture Notes in Computer Science. Vol. 7979, pp. 344–357, 2013

- Nobile M.S., Besozzi D., Cazzaniga P., Mauri G., Pescini D.
A GPU-Based Multi-Swarm PSO Method for Parameter Estimation in Stochastic Biological Systems Exploiting Discrete-Time Target Series
In 10th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Computational Biology, EvoBIO 2012, Malaga (Spain), Proceedings. M. Giacobini, L. Vanneschi, and W. Bush (Eds.). Lecture Notes in Computer Science. Volume 7264, pp. 74–85, 2012

Book chapters

- Nobile M.S., Cipolla D., Cazzaniga P., Besozzi D.
  GPU-powered Evolutionary Design of Mass-action Based Models of Gene Regulation
  In *Evolutionary Algorithms in Gene Regulatory Network Research*, H. Iba, N. Noman (Eds.), John Wiley & Sons, (2014, in press)

Posters and abstracts

- Besozzi D., Nobile M.S., Cazzaniga P., Cipolla D., Mauri G.
  From the Inference of Molecular Structures to the Analysis of Emergent Cellular Dynamics: Accelerating the Computational Study of Biological Systems with GPUs
  Proceedings of the NETTAB 2014 Workshop: from Structural Bioinformatics to Integrative Systems Biology, Torino (Italy), October 15-17, pp. 88–90, 2014

- Bellini M., Besozzi D., Cazzaniga P., Mauri G., Nobile M.S.
  Simulation and Analysis of the Blood Coagulation Cascade Accelerated on GPU
  Proceedings of 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2014), Torino (Italy), published by IEEE, pp. 590–593, 2014

- Caravagna G., Cazzaniga P., Nobile M.S., Pescini D., Re A.
  Enhancing Simulation of Chemical Reactions at Mesoscales
  BITS 2014 - 11th Annual Meeting of the Bioinformatics Italian Society, Rome (Italy), February 26-28, 2014

- Besozzi D., Caravagna G., Cazzaniga P., Nobile M.S., Pescini D., Re A.
  GPU-powered Simulation Methodologies for Biological Systems
  In A. Graudenzi, G. Caravagna, G. Mauri, M. Antoniotti (Eds.): Wivace 2013 Italian Workshop on Artificial Life and Evolutionary Computation, Electronic Proceedings in Theoretical Computer Science, pp. 93–97, 2013

- Besozzi D., Cazzaniga P., Colombo R., Mauri G., Nobile M. S., Pescini D.
  Accelerating the Computational Analysis of Biological Systems by Means of Graphics Processing Units

ICSB 2013 - 14th International Conference on Systems Biology, Copenhagen (Denmark), 2013

- Besozzi D., Colombo R., Cazzaniga P., Nobile M. S., Pescini D., Mauri G.
  A GPU-powered Computational Analysis of PCNA Ubiquitylation Processes Involved in UV-induced DNA Lesions Bypass
  ICSB 2013 - 14th International Conference on Systems Biology, Copenhagen (Denmark), 2013

- Cazzaniga P., Colombo R., Nobile M.S., Pescini D., Mauri G., Besozzi D.
  GPU-powered Sensitivity Analysis and Parameter Estimation of a Reaction-based Model of the Post Replication Repair Pathway in Yeast
  Proceedings of the 10th International Workshop on Computational Systems Biology (WCSB 2013), Tampere (Finland), p. 109, 2013

- Nobile M.S., Besozzi D., Cazzaniga P., Mauri G., Pescini D.
  Estimating Reaction Constants in Stochastic Biological Systems with a Multi-swarm PSO Running on GPUs
  Proceedings of the 14th International Conference on Genetic and Evolutionary Computation Conference Companion. ACM New York, NY, USA, GECCO Companion '12, pp. 1421–1422, (2012)

Part I

Theoretical Background

Chapter 2

Modeling and simulation of biochemical systems

According to the system-level perpective of Systems Biology [8], in order to have a general outlook of the phenomenon of interest it is useful to represent the biological system by means of a proper mathematical formulation, able to integrate the different kinds of data obtained from laboratory experiments. An essential advantage of mathematical models is that they can help to formulate *in vivo*-testable hypotheses. Therefore, there is the need for models allowing the analysis of regulatory features, giving predictive power to post-genomic data.

2.1 Modeling approaches

Biochemical systems can be described at different levels of detail, and their properties can be simulated and analyzed using a variety of computational techniques according to the purpose of the study, starting from the identification of a specific biological problem that requires — or might benefit from — a complementary analysis based on experimental investigations, mathematical modeling and computational methods. The scientific question that motivates the development of a model represents the essential key in designing the most appropriate modeling and computational workflow. The choice of the proper modeling technique depends on the following factors:

- the purpose of the model and the information that we expect to collect from *in silico* analyses;
- the availability and quality of data about the components of the biochemical system and their interactions.

These factors implicitly determine the proper level of abstraction for the model to be defined. In particular, they will bring to the definition of coarse-grained (e.g., *interaction-based* or *constraint-based*) or fine-grained (e.g., *mechanism-based*) models.

Interaction-based and constraint-based models can facilitate the identification of pivotal components or modules of the system under investigation, although they neglect most of the quantitative and kinetic properties of its components and interactions (e.g., biochemical reactions). The mechanism-based modeling, on the other hand, has the greatest predictive capability concerning the functioning of the system at molecular level, but has limited applicability since it requires detailed kinetic information about the interaction between the components of the system. A schematic overview of the main modeling approaches (interaction-based, constraint-based, mechanism-based) and their respective features is given in Figure 2.1.

The interaction-based approach relies on the definition of an interaction graph in which the nodes represent the chemical species and the edges connect two nodes that are known to have some kind of interaction. Such methodology allows a large-scale modeling of a biochemical systems (e.g., *genome-wide*) thanks to the development of high-throughput technologies able to generate an unprecedented wealth of quantitative data about living systems [317]. Despite this remarkable extension and complexity, the analysis of interaction graphs is generally limited to topological characteristics like degree distribution, clustering coefficient, shortest paths or network motifs. This type of analysis allows to determine the main features of the structural organization of the network at the large-scale level, to understand the underlying processes at the basis of the evolution of the structure itself [184] and can provide insights about the sources of robustness and redundancy in the biochemical network [21].

A further class of modeling techniques, conceptually close to interaction-based models, is represented by *Bayesian Networks* (BANs) [251]. BANs are directed acyclic graphs, whose vertexes represent random variables and arcs represent conditional dependencies. Altough BANs are powerful means to represent probabilistic relationships between actors of a biochemical system (e.g., genes and their mutual regulations), their applicability is limited because they do not allow to model loops and feedback mechanisms [35].

By including additional information to the bare network structure it is possible to build *constraint-based models*, which permit the exploration of a the set of allowable flux distributions (i.e., flows of metabolites [248]). Stoichiometry is the easiest information that can be added (see Section 2.2.1 for further information), although by itself may not be enough to fully determine the feasible states of the system. Further constraints used

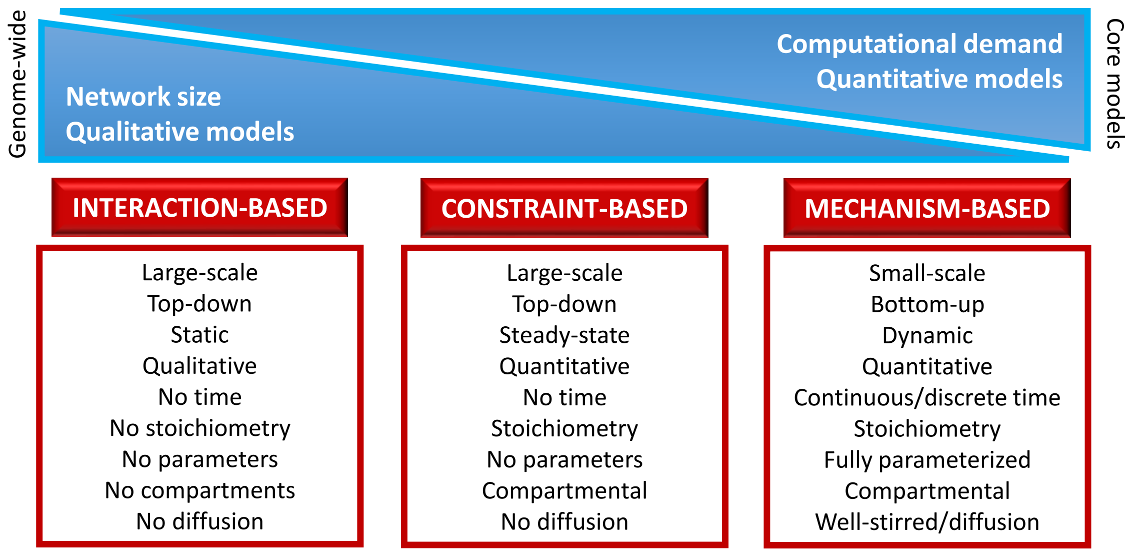


Figure 2.1: Schematic overview of the main modeling approaches for biological systems, together with their principal characteristics and differences. Moving from the coarse-grained (*interaction-based*, *constraint-based*) to the fine-grained (*mechanism-based*) approach, models vary with respect to: (*i*) the *size* of the system, defined in terms of the number of components and respective interactions included in the model, which decrease from genome-wide to core models; (*ii*) the *computational costs* required for the analysis of the model, which increase from the analysis of the topological properties of the network typical of interaction-based models, to the study of flux distributions typical of constraint-based models, to the investigation of the system dynamics typical of mechanism-based models; (*iii*) the nature of the computational results together with the predictive capability, which changes from *qualitative* to *quantitative* while moving from interaction-based models (characterized by a high level of abstraction) to mechanism-based models (fully parameterized and describing the system at the level of the functional chemical interactions). Figure adapted from [50].

to limit the space of solutions may include thermodynamic constraints (e.g., regarding the reversibility of a reaction) or transcriptomic and enzyme capacity constraints. Various techniques allow to investigate the feasible flux distributions at steady-state within a network, including Extreme Pathway Analysis [267, 301], Elementary Mode Analysis [335], Flux Balance Analysis [248]. Since the constraints on fluxes are generally linear equations, the problem of identifying the most feasible or optimal state can be formulated as a Linear Programming problem and solved using one of the available computational approaches, like the Simplex Method described in Chapter 3.

Finally, mechanism-based models describe the system at the level of functional biochemical interactions, thus they have a higher predictive capability with respect to the other modeling approaches. For this reason, mechanistic modeling is considered the most likely candidate to achieve a detailed comprehension of biological systems [50], since it can lead to quantitative predictions of cellular dynamics, thanks to its capability to reproduce the temporal evolution of all molecular species occurring in the model. Nonetheless, the computational complexity of the simulation and analysis of such models increases with the size (in terms of components and interactions) of the systems, limiting the feasibility of this approach. Moreover the usual lack of quantitative parameters (e.g., kinetic constants, initial molecular concentrations of the species) and the partial lack of knowledge about the molecular mechanisms, sometimes due to the difficulty or impossibility to perform *ad hoc* experiments, represent further limits to a wide applicability of this modeling approach. All these problems — simulation performances, parameter estimation, and reverse engineering — are among the subjects of the present thesis work.

A mechanistic model of a biochemical system can be defined by using various mathematical formalisms, for instance reaction-based models, differential equations, S-systems, and Petri nets. All these formalisms, and the relative simulation techniques, will be described in detail in the following sections.

2.2 Mechanistic modeling

In this thesis, given a biological system denoted by Ω, its biochemical reactions are assumed to obey the mass-action kinetics (MAK) [59], except where stated otherwise. MAK is a fundamental and empirical law governing biochemical reaction rates: it states that, in a diluted solution, the rate of an elementary reaction (i.e., a reaction with a single mechanistic step) is proportional to the product of the concentration of its reactants raised to the power of the corresponding stoichiometric coefficient [59, 226]. MAK is the most general framework for the description of biochemical kinetics, and it represents the basis of the modeling, simulation and inference tools presented in this thesis. In addition, since no diffusion processes will be considered in what follows, the biochemical systems are assumed to be well-stirred, at thermal equilibrium and characterized by a fixed volume.

2.2.1 Reaction-based models

Given a biochemical system Ω, a mechanistic reaction-based model (RBM) is defined by specifying a set of N chemical species $\mathcal{S} = (S_1, \dots, S_N)$, involved in a set of M

chemical reactions $\mathcal{R} = (R_1, \ldots, R_M)$ of the form:

$$R_j : \alpha_{j1} \cdot S_1 + \ldots + \alpha_{jN} \cdot S_N \xrightarrow{k_j} \beta_{j1} \cdot S_1 + \ldots + \beta_{jN} \cdot S_N, \tag{2.1}$$

where $\alpha_{ji}, \beta_{ji} \in \mathbb{N}$ are the stoichiometric coefficients of R_j, and $k_j \in \mathbb{R}^+$ is the kinetic constant associated to R_j. The species occurring on the left-hand (right-hand) side of R_j are called reagents (products, respectively). This formalization defines a network η of interacting chemical species.

Reactions $R_1, \ldots, R_M$ implicitly define two matrices, $\mathbf{MA}, \mathbf{MB} \in \mathbb{N}^{M \times N}$, having α_{ji} and β_{ji} as elements, respectively. $x_i(t) \in \mathbb{N}$ denotes the number of molecules of species S_i present in Ω at time t, so that $\mathbf{x} = \mathbf{x}(t) \equiv (x_1(t), \ldots, x_N(t))$ represents the state of the system at time t. $\mathbf{MV}$ denotes the state change matrix associated to system, defined as $\mathbf{MV} \equiv \mathbf{MB} - \mathbf{MA}$. Each row of this matrix, $\mathbf{MV}_j \equiv \boldsymbol{\nu}_j \equiv (\nu_{j1}, \ldots, \nu_{jN})$, is a state change vector that consists of elements $\nu_{ji} = \beta_{ji} - \alpha_{ji}$, $\nu_{ji} \in \mathbb{Z}$, representing the stoichiometric change of species S_i due to reaction R_j.

Besides $\mathbf{MV}$, it is possible to define a supplementary state change matrix $\overline{\mathbf{MV}}$, where $\bar{\nu}_{ji} = 0$ for each $S_i \in \mathcal{F}$, for a given $\mathcal{F} \subset \mathcal{S}$ that is composed by the molecular species whose amounts do not change over time; the subset $\mathcal{F}$ can be used to model a continuous "feed" of molecules into the system. This condition is useful to mimic, for instance, the non-limiting availability of some chemical resources, or the execution of *in vitro* buffering experiments, in which an adequate supply of some species is introduced in Ω in order to keep their quantity constant [58].

If $\alpha_{ji} = 0$ for all $i = 1, \ldots, N$, then R_j is called a *source* reaction (denoted as $\emptyset \rightarrow$ *products*); on the contrary, if $\beta_{ji} = 0$ for all $i = 1, \ldots, N$, then R_j is called a *sink* or degradation reaction (denoted as *reagents* $\rightarrow \emptyset$). Obviously, reactions in the form $\alpha_{ji} = 0$ and $\beta_{ji} = 0$ for all $i = 1, \ldots, N$ (that is, reactions equivalent to $\emptyset \rightarrow \emptyset$) are not considered. Reactions of the form $R_j : \alpha_i S_i \rightarrow \beta_i S_i$, for any α_i and β_i, are considered meaningless, since they correspond to unfeasible biochemical processes where α_i molecules of species S_i are converted into β_i molecules of the same species.

The amount of chemicals occurring in a RBM can be given either as numbers of molecules or as concentrations. RBMs are generally exploited for the stochastic modeling of biological systems and, in such a case, the state of the system is represented by a vector of integer values corresponding to the exact molecular amount of chemical species. In these models, the value k_j in Equation 2.1 is generally denoted by c_j and represents the stochastic constant associated to the reaction, that is, a real valued parameter encompassing the physical and chemical properties of R_j [100]. The fundamental hypothesis of the stochastic formulation of chemical kinetics states that the average probability of reaction R_j to occur in the interval $(t, t + \mathrm{d}t)$ is exactly $c_j \mathrm{d}t$. The probabilities of all reactions can be exploited to calculate a trajectory of the system, using one of the stochastic simulation algorithms described in Section 2.3.2. Appendix A contains examples of biochemical models formalized by means of RBMs.

2.2.2 Chemical Master Equation

If the biochemical system Ω is investigated by means of a stochastic approach, it can be modeled with the Chemical Master Equation (CME), which describes the probability distribution function associated to Ω [339].

Specifically, the probability $P(\mathbf{x}, t|\mathbf{x}_0, t_0)$, i.e., the probability that the system will be in state $\mathbf{x}$ at time t starting from state $\mathbf{x}_0$ at time t_0, can be calculated by means of the following Master Equation:

$$\frac{\partial P(\mathbf{x}, t|\mathbf{x}_0, t_0)}{\partial t} = \sum_{j=1}^{M} [a_j(\mathbf{x} - \boldsymbol{\nu}_j) P(\mathbf{x} - \boldsymbol{\nu}_j, t|\mathbf{x}_0, t_0) - a_j(\mathbf{x}) P(\mathbf{x}, t|\mathbf{x}_0, t_0)],$$

where $\boldsymbol{\nu}_j$ is the state change vector of reaction R_j formalized in the previous section.

Even though the CME allows an exact derivation of the probability $P(\mathbf{x}, t)$ for an arbitrary couple of initial state and time t, its analytical solution is generally untractable (see Section 2.3.2).

2.2.3 Differential equations

The traditional mechanistic modeling and simulation approach consists in defining a system of coupled Ordinary Differential Equations (ODEs), which describe the rate of change of each chemical species involved in the system. For instance, the following reversible reaction, which involves three chemical species S_1, S_2 and S_3

$$S_1 + S_2 \underset{k_r}{\overset{k_f}{\rightleftharpoons}} S_3 \tag{2.2}$$

can be modeled, by assuming MAK, as the following system of coupled ODEs:

$$\begin{aligned}
\frac{\mathrm{d}[S_1]}{\mathrm{d}t} &= -k_f[S_1][S_2] + k_r[S_3] \\
\frac{\mathrm{d}[S_2]}{\mathrm{d}t} &= -k_f[S_1][S_2] + k_r[S_3] \\
\frac{\mathrm{d}[S_3]}{\mathrm{d}t} &= k_f[S_1][S_2] - k_r[S_3]
\end{aligned}$$

where $[\cdot]$ denotes the concentration of the chemical species.

ODE models in the form described above can be generalized by means of *Reaction Rate Equations* (RREs) [14]. Let $S_i(t)$ denote the concentration of the i-th species at time t. If we assume M reactions involving N chemical species, then

$$\frac{\mathrm{d}S_i}{\mathrm{d}t} = \sum_{j=1}^{M} \nu_{ji} a_j(\mathbf{S}, k_j), \text{ for } i = 1, \dots, N,$$

where a_j is the *propensity* of reaction R_j, a function of reactants concentrations (contained in the state vector $\mathbf{S}$) and the kinetic parameters of the reaction itself. For instance, in the case of Equation 2.2, the propensity of the forward reaction according to the MAK is equal to $-k_f[S_1][S_2]$, since it involves S_1 and S_2 as reactants. It is worth noting that ODE-based models are not limited to MAK, but can exploit different kinetics like Hill functions, Michaelis-Menten kinetic [226], or any arbitrary mathematical function describing a plausible kinetic behavior.

Even though ODEs provide a sound mathematical framework, they represent an approximation of the system, since their simulation (performed using numerical integration algorithms) does not consider the stochasticity that usually characterizes biochemical systems [355]. Indeed, a fundamental aspect that should be considered in the definition of mathematical models of biological systems concerns the experimental evidences that cellular regulation networks — in particular those characterized by low quantities of some molecular species — are often affected by noise [85]. The randomness at the molecular scale can induce stochastic phenomena at the macromolecular scale, leading to non deterministic behaviors. The classical approaches relying on ODEs are not suitable to capture the effects of stochastic processes; in this context, a typical example regards the bistability phenomenon (one example is provided in Figure 2.2), that can be effectively investigated by means of stochastic approaches [115].

Stochastic Differential Equations (SDEs), like the Chemical Langevin Equations (CLEs) [103], allow to explicitly extend the rate equations with noise terms, in order to produce simulated dynamics of the system that are more adherent to the biological reality.

CLEs themselves represent an approximation of the dynamics of the system that is feasible for situations where the concentration of chemical species, given a specific reaction volume, roughly corresponds to a large number of molecules. In such a case, the time evolution of the system is modeled as

$$\mathbf{x}(t+\tau) = \mathbf{x}(t) + \sum_{j=1}^{M} \boldsymbol{\nu}_j \mathcal{N}(a_j(\mathbf{x}(t))\tau, a_j(\mathbf{x}(t))\tau),$$

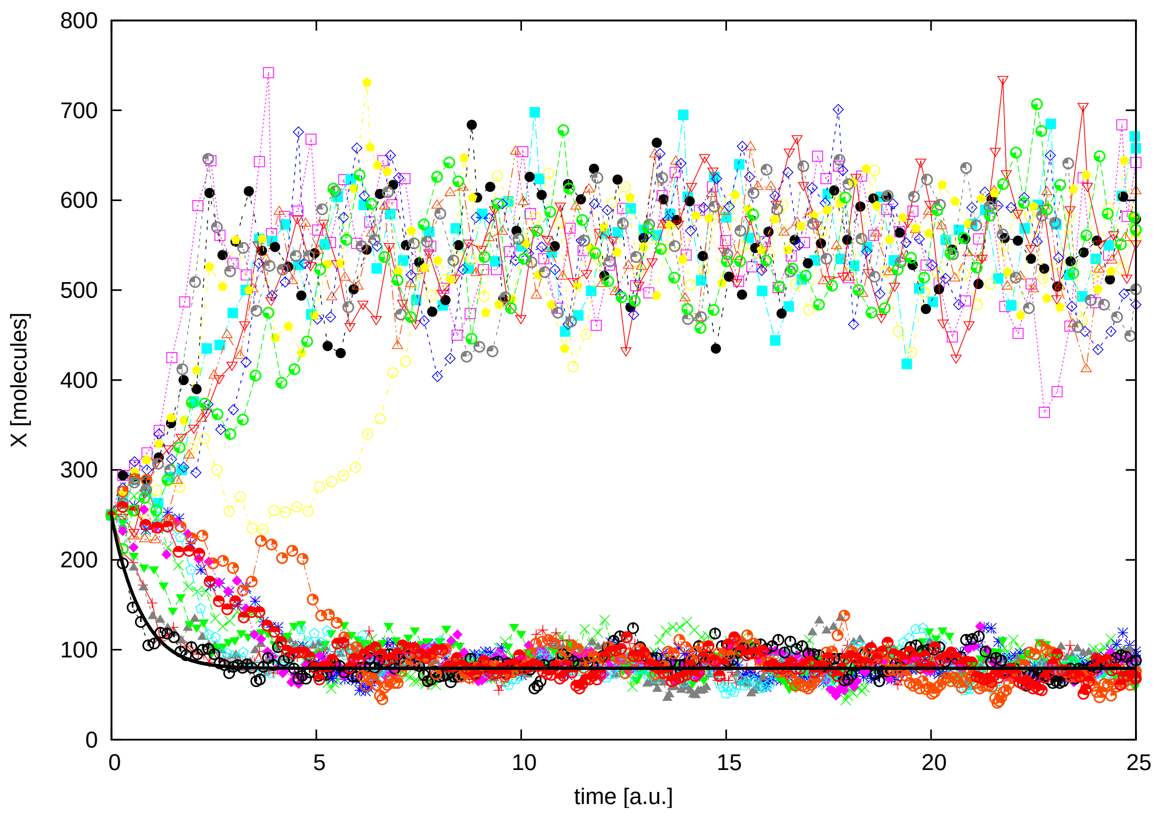


Figure 2.2: Example of the non-suitability of ODE-based simulation in the case of systems characterized by bistability. The figure compares the outcome of a deterministic simulation (black line) and the dynamics of stochastic simulations (colored dotted lines) of the Schlögl model (see Appendix A). This simple biochemical system is characterized by bistability, that is, it has two equilibrium states which can be reached from the same initial condition. Whilst stochastic simulation correctly reproduces this phenomenon, ODEs can only collapse into one of the two states, starting from the chosen initial condition..

where $\mathbf{x}(t)$ denotes the state of the system at time t (e.g., the vector of species concentrations) and $\mathcal{N}(\mu, \sigma^2)$ corresponds to a normally distributed random variable with mean μ and variance σ^2.

When the number of reactions occurring in a time interval τ is small, or the amount of molecules is in the order of hundreds or less, CLEs are no longer meaningful and reliable. Different approaches should be preferred, like the RBMs described in Section 2.2.1.

2.2.4 S-systems

S-systems represent an approximate ODE-based modeling formalism which exploits power-law functions, so that a biochemical system is represented a system of coupled non-linear differential equations [297] in the form:

$$\frac{\mathrm{d}S_i}{\mathrm{d}t} = \Phi_i \prod_{j=1}^{N} S_j^{g_{ij}} - \Psi_i \prod_{j=1}^{N} S_j^{h_{ij}},$$

where N is the number of components and S_i is the amount of the i-th chemical species or component of the biological system; Φ_i and Ψ_i are the rate constants. Terms g_{ij} (h_{ij}) are the *kinetic orders* representing the strength of the influences that increase (decrease) the amount of S_i, respectively.

These models can provide a good compromise between accuracy and mathematical flexibility: despite the simplifications, S-systems are still rich enough to capture the dynamics and the mechanisms of biochemical systems [239]. They are largely exploited for the modeling and investigation of gene expression [239, 347]: in such models, S_i denotes the expression level of gene i, while strengths represent the regulation exerted by the rest of the gene network on gene i.

2.2.5 Petri nets

Petri Net (PN) is a modeling formalism introduced by Petri [256] for the modeling of distributed, asynchronous and concurrent systems [223]. A PN is defined as a weighted, directed, bipartite graph consisting of two kinds of nodes: the nodes representing the state (or conditions) of the system, called *places* and denoted by circles, and the nodes representing *transitions* (or events) between places, denoted by rectangles. Nodes are

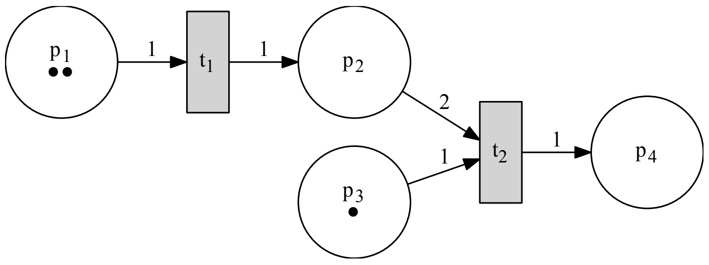


Figure 2.3: A Petri Net composed of four places (p_1, p_2, p_3, p_4) and two transitions (t_1, t_2). The weights on the arcs from places to transitions represent the minimum number of tokens required for the transition to be enabled. Since place p_1 contains 2 tokens while p_3 contains 1 token, only transition t_1 is enabled and available for firing. On the contrary, the pre-condition for transition t_2 is not verified, because at least 2 tokens are needed in place p_2. The weight on the arcs from a transition (place) to a place (transition) represent the number of tokens that will be added to (subtracted from) that place as a consequence of the firing of the transition.

interconnected by directed arcs; an arc can only connect a place to a transition and vice versa, i.e., arcs cannot connect two places or two transitions. In standard place-transition PNs, each place can contain a discrete and positive number of marks, which are called *tokens*. Figure 2.3 shows an example of a PN with four places (p_1, p_2, p_3, p_4) and two transitions (t_1, t_2). Places p_1 and p_3 have two and one tokens, respectively.

PNs are extensively exploited for the simulation and analysis of the structural and behavioral properties of complex systems. Basic notions and notations of PNs can be found in [191]. Formally, a PN is a 5-tuple (P, T, F, W, M_0) where:

- $P = \{p_1, \ldots, p_m\}$ is a finite set of *places*;
- $T = \{t_1, \ldots, t_n\}$ is a finite set of *transitions*, such that $P \cap T = \emptyset$;
- $F \subseteq (P \times T) \cup (T \times P)$ is the set of *arcs*;
- $W : F \to \mathbb{N}$ is a *weight* function, which associates a non-negative integer value to each arc;
- $M_0 : P \to \mathbb{N}$ is the initial state of the net, called the *initial marking*.

The marking $\mathbf{M} \in \mathbb{N}^m$ of a PN represents its state and M_{p_i} is the number of *tokens* in place $p_i \in P, i = 1, \ldots, m$. A place $p \in P$ is said *input* of a transition $t \in T$ if there is an arc from p to t, that is, $(p, t) \in F$; $p \in P$ is said *output* of $t \in T$ if there is an arc from t to p, that is, $(t, p) \in F$.

The *preset* (*postset*) of a transition t is the set of its input places and is denoted as ${}^\bullet t = \{p \in P | (p, t) \in F\}$ (respectively, $t^\bullet = \{p \in P | (t, p) \in F\}$). Presets and postsets are similarly defined for places: ${}^\bullet p = \{t \in T | (t, p) \in F\}$ and $p^\bullet = \{t \in T | (p, t) \in F\}$. When PNs are used to model real systems, transitions represent possible events that involve the connected places. In this sense, places represent conditions for the firing of transitions and, in particular, the input places are called pre-conditions, whilst the output places are called post-conditions.

A transition $t \in T$ is *enabled* if its pre- and post-conditions are verified. The pre-condition for t is verified if, for each input place $p \in {}^\bullet t, M_p \geq w(p, t)$, where $w(p, t)$ is the weight of the arc from p to t. Symmetrically, the post-condition for t is verified if, for each output place $p \in t^\bullet, M_p + w(t, p) \leq K(p)$, where $K(p)$ is called the *capacity* of place p. A PN in which there are no post-conditions is referred to as an infinite capacity net [223].

When a transition is enabled, it can *fire*. When a transition is fired, it changes the amount of tokens inside both input and output places according to the transition rule: $w(p, t)$ tokens are removed from each $p \in {}^\bullet t$, while $w(t, p')$ tokens are added to each

$p' \in t^\bullet$. The marking $\mathbf{M}$ of a PN changes in time as a consequence of the firing of transitions, which happens in a complete non-deterministic fashion: any of the enabled transitions can fire in any moment, stated that each firing is a completely atomic event.

Because of their bipartite graph structure, PNs offer an ideal conceptual framework for the modeling of biochemical networks [53] defined as a set of reactions in the form of Equation 2.1. To this aim, a transformation of a network η into a corresponding PN (and vice versa) needs to be defined. Briefly, a mapping $\psi_{\mathcal{S}} : \mathcal{S} \rightarrow P$ can be used to associate the species to the places of a PN, where the transitions represent the chemical reactions (i.e., $\psi_\mu : \mathcal{R} \rightarrow T$) and where the weights correspond to the stoichiometry of the reagents and products of each reaction (i.e., $\psi_{reac} : \alpha_{ji} \rightarrow W(p_i, t_j)$, $\psi_{prod} : \beta_{ji} \rightarrow W(t_j, p_i)$). According to these mappings, it is straightforward to show that the PN on the left in Figure 2.3 represents the following set of reactions:

- $R_1 : S_1 \rightarrow S_2$;
- $R_2 : 2S_2 + S_3 \rightarrow S_4$.

Extensions and Properties of PNs

PNs can be easily extended to support continuous places [10], stochastic behavior [152] and functional transitions [206]. Thus, they represent a versatile approach for the modeling of a wide spectrum of biochemical systems, allowing both qualitative and quantitative analysis [194]. In addition, it is possible to exploit traditional PN analysis methods to determine properties of a PN with a specific biological interpretation [53]. Examples of such properties are:

- *boundedness*: independently from the assignment of M_0, the number of tokens in one place p is limited to a maximum value: in a metabolic network this means that metabolite p cannot accumulate;
- *liveness*: a transition t is *live* if it can always fire, meaning that the associated reaction can always occur in the system. A PN is said *live* if all transitions are *live*;
- *deadlock freeness*: a PN has no deadlocks if in any reachable marking $\mathbf{M}$ at least *one* transition can fire;
- *reachability* of a marking $\mathbf{M}$: there exists a time evolution of the system from M_0 to $\mathbf{M}$. This means that a specific state of the biochemical system can be reached from the initial conditions;

- *P-invariants* are sets of places whose weighted sum of tokens remains constant throughout the evolution of the net. These sets can be interpreted as conservation relations;

- *T-invariant*: a firing sequence that does not change the marking. They correspond to cyclic behaviors of the system.

Both liveness and deadlock-freeness are structural properties, even though they depend on the initial marking M_0. One example is the glycolytic pathway, in which two ATP molecules that are involved in the upstream reactions of the metabolic cascade are responsible for the production of four ATP molecules downstream the whole pathway. Such biochemical system is deadlock-free as long as in the initial marking there are at least two tokens in the place associated to ATP [366]. In general, the same consideration applies to any auto-catalytic reaction, which cannot be enabled without any initial tokens.

2.2.6 Other mechanistic modeling methods

For the sake of completeness, additional mechanism-based modeling approaches are reviewed in this section, although not employed in this thesis.

A first extension of RBMs is the *rule-based* modeling, which can be exploited to overcome the combinatorial complexity due to multiple protein-protein interactions, which can cause an explosion of the number of intermediate complexes and chemical reactions [134]. In the rule-based approach, molecules are represented by objects with extended characteristics, so that reactions can be defined as rules on patterns of objects instead of specific chemical species, reducing the complexity of the model [135]. Rule-based models can be simulated by re-expanding the system by means of an explicit enumeration of all the chemical species and reactions [207], which can be huge and difficult to simulate. Alternatively, they can be simulated by using "network-free" Monte Carlo methods [362] which explicitly represent every molecule that is present in the system. Of course, the latter approach is not feasible for systems characterized by a large number of molecules, because of its prohibitive spatial complexity. A possible solution to this problem is the use of "hybrid" particle-counters approach, in which the chemical species that exist in large numbers are represented by normal variables instead of particles [60].

A further extension of the approaches described so far is the *spatial* modeling, which no longer considers the system as well-stirred but explicitly models reaction-diffusion mechanisms happening inside the reaction volume. Spatial models can be realized by means of differential equations. More specifically, *Partial Differential Equations* (PDEs) are exploited to model the diffusion of molecules into a finite number of subvolumes, which collectively represent the reaction space of the biochemical system Ω. It is important to underline that the smaller the subvolumes, the more realistic but computationally expensive the simulation. PDEs are strictly deterministic, thus they do not allow to consider the intrinsic stochasticity of cellular systems, which has deep implications when then the amount of molecules is very small (e.g., transcription factors in the nucleus).

In spatial stochastic modeling, the state of the system is defined by the copy number of each chemical species in each subvolume, and the state change considers both the chemical reactions (transforming the species) and the diffusion reactions between neighboring volumes. Simulation of spatial stochastic models can be performed by algorithms like the *Next Subvolume Method* (NSM) [83].

Reaction-diffusion systems can also be modeled by means of *Cellular Automata* (CA) [350]. In this discrete-time/discrete-space formalism, reactions and diffusion mechanisms are modeled as sets of probabilistic update rules, which describe the update of the state of each automaton (e.g., a vector of molecules amounts) according to its current state and the state of its neighbors. Although CA have been used to perform quantitative investigations [356], lattice size and neighborhood geometry (e.g., square, hexagonal, trigonal) can have a relevant impact to the outcome of the simulation [310] and make this approach difficult to validate.

Another approach for the formalization of spatial systems is *Agent-Based* modeling (ABM). In such methodology, each molecule in the system is explicitly represented by an agent, which is characterized by a well determined spatial positioning. According to a set of rules, agents move in the space; the proximity to other agents causes their interaction and the change of state of the system. ABM can be unsuitable for the simulation of large-scale biochemical systems, since the computational complexity largely increases with the number of molecules. Moreover, since ABM is based on hypothetical rules of interaction, they may not represent the best option to investigate patterns and analyze emergent properties [12].

Modeling by means of π-calculus is an alternative mechanistic non-spatial methodology, based on *Process Algebra* [215], a formal language originally developed to model concurrent computational systems exchanging messages (e.g., mobile telephones). In this formulation, processes represent molecules and communication defines chemical interactions. Regev *et al.* showed that π-calculus is a suitable methodology for modeling various biochemical mechanisms [275] (e.g., transcriptional mechanisms, signal transduction, metabolic pathways).

A completely different modeling approach — which neglects both the spatiality and the dynamics of the system — is represented by *Boolean Networks* (BONs). This formalism, introduced by Kauffman for the modeling of gene expression networks [154], relies on a discrete and finite set of boolean variables (e.g., a gene and its activation state), each of which has a boolean activation function taking as input the state of a subset of the other variables (e.g., the activation states of other genes). This description defines the topology of a graph, in which nodes are the variables and arcs describe the input connections. The massively parallel and synchronous update of the variables, according to the respective activation functions and input states, allows to simulate the system. Even though the possibility to set a gene's state to *false* allows to perform *in silico knock-out* experiments, the outcome of the simulation of BONs is not quantitative. Moreover, given a BON characterized by N nodes, the space of the possible states is 2^N, so that the dynamics of the simulation is necessarily periodic. Nevertheless, BONs allow the large-scale investigation of biochemical systems [4] from the points of view of both graph and dynamic systems theories (e.g., attractors, basins of attraction, reachability).

2.3 Simulation methods

Once a biochemical system is modeled using one of the aforementioned *mechanistic* approaches, its dynamic behavior can be investigated by using a simulation methodology that is associated to the chosen modeling formalism.

It is worth outlining here that, in order to perform the simulation of a mechanistic model, it is not sufficient to describe the system as a simple interaction network. The simulation techniques (both deterministic and stochastic) described in the following sections assume that a proper **kinetic parameterization** and the **initial state** of the system are known, and specified as part of the model.

2.3.1 Deterministic simulation

The simulation of biochemical systems modeled by systems of ODEs can be performed by means of an ODE solver. By providing an initial state of the system, along with a set of kinetic constants for the ODEs, it is possible to describe the temporal evolution of the system by using any of the existent ODEs numerical integration algorithms.

The most straightforward numerical algorithm for solving ODE systems is Euler's method (EM), an iterative algorithm defined by mathematician Euler in 1768 [41]. In EM, the differential equation is seen as a formula to calculate the slope of the tangent line to the unknown curve defined by the set of ODEs. More precisely, considering $y'(t) = f(t, y(t))$, given the initial point $y(t_0)$ and a step size Δ, we can exploit the ODE to calculate the slope of the curve in each point starting from the previous point:

$$y_{n+1} = y_n + \Delta f(t_n, y_n),$$

where $y_n = y(t_n)$. Euler's method can be derived from the Taylor expansion of a function y around a point t_0, ignoring the quadratic and higher-order terms:

$$y(t_0 + \Delta) = y(t_0) + \Delta y'(t_0) + \frac{1}{2}\Delta^2 y''(t_0) + O(\Delta^3).$$

Thus, EM is a first-order approximation method, whose error at a given time t is $O(\Delta^2)$ (Figure 2.4).

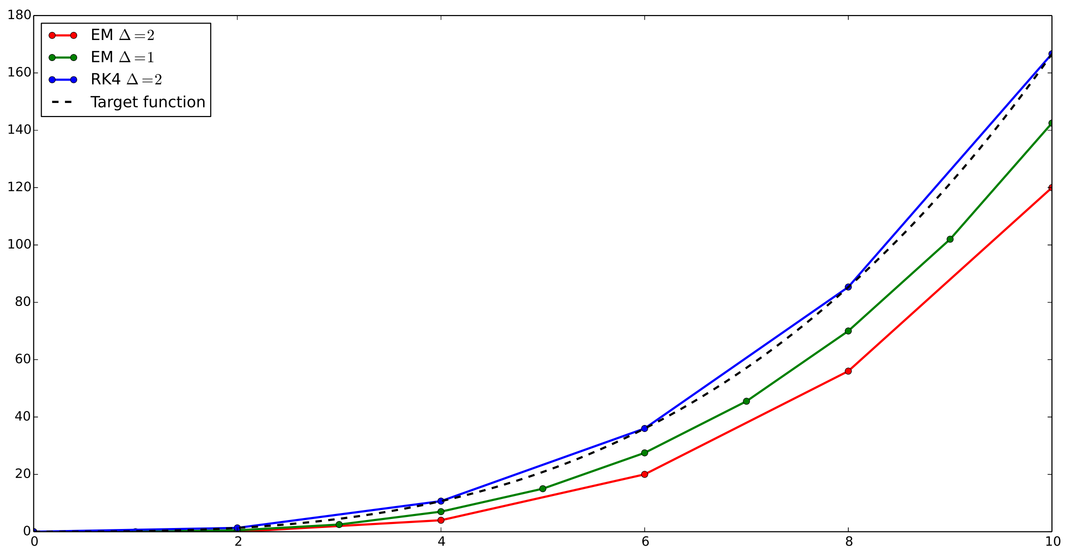


Figure 2.4: Visualization of the error of EM and RK4 methods due to the approximation, when solving $y' = y^2/2$, $y(0) = 0$. The two curves of EM were calculated with $\Delta = 2$ (red line) and $\Delta = 1$ (green line); RK4 uses $\Delta = 2$ (blue line). In the EM, the error is $O(\Delta^2)$, so that the approximation improves as the time step gets smaller and smaller, with a computational complexity scaling proportionally with Δ. In the case of RK4, even with $\Delta = 2$, the error is smaller than EM, representing a better approximation of the target curve.

Runge-Kutta methods

In order to mitigate the error of EM, at the beginning of 1900 mathematicians Runge and Kutta introduced a family of methods extending the idea at the basis of EM [41]. One example is the *midpoint method*, which performs a two-stage calculation:

$$y_{n+1} = y_n + \Delta f\left(t_n + \frac{1}{2}\Delta, y_n + \frac{1}{2}\Delta f(t_n, y_n)\right). \tag{2.3}$$

In Equation 2.3, the slope is calculated twice: the first time to determine the slope of the curve from t_n to $t_n + \Delta$; the second time, this value is exploited to calculate the slope at the midpoint $t_n + \frac{1}{2}\Delta$, which is finally used to determine the value of point y_{n+1}. This modified EM gives a final error of order $O(\Delta^2)$.

The most popular Runge-Kutta algorithm is called *RK4*. RK4 is an extension of the midpoint method and it is based on the calculation of the following quantities:

$$\begin{aligned}
h_1 &= f(t_n, y_n) \\
h_2 &= f(t_n + \tfrac{\Delta}{2}, y_n + \tfrac{\Delta}{2}h_1) \\
h_3 &= f(t_n + \tfrac{\Delta}{2}, y_n + \tfrac{\Delta}{2}h_2) \\
h_4 &= f(t_n + \Delta, y_n + \Delta h_3).
\end{aligned}$$

Then, the new point y_{n+1} is calculated as the weighted average of quantities h_1, h_2, h_3 and h_4:

$$y_{n+1} = y_n + \frac{\Delta}{6}(h_1 + 2h_2 + 2h_3 + h_4).$$

RK4 is a fourth-order method whose final error is $O(\Delta^4)$, which produces an improved approximation of the unknown curve with respect to EM (Figure 2.4).

LSODA

Despite the remarkable quality of the derived solutions and its implementation simplicity, RK4 may be unfit for the simulation of some classes of biochemical systems. Specifically, it is not efficient for models characterized by *stiffness*, that is, two well-separated dynamical modes, determined by fast and by slow reactions, the fastest of which is stable [105].

To date, one of the most efficient algorithms to integrate a system of ODEs is the Livermore Solver of Ordinary Differential Equations (LSODA) [257], a solver able to automatically recognize stiff and non-stiff systems, and to dynamically switch between the most appropriate integration procedure: Adams' method in the absence of stiffness, and the Backward Differentiation Formulae (BDF) otherwise. In particular, LSODA initially assumes a non-stiff problem and dynamically monitors data in order to switch to BDF. Since the BDF exploits the Jacobian matrix of the ODE system, LSODA requires the user to calculate it and encode the matrix as a function, using LSODA's implementation in one available programming language (e.g., FORTRAN or C).

LSODA's implementation programming interface allows to provide to the algorithm several functioning settings, in order to control the performance and quality of ODEs integration. In particular, it is possible to set the absolute and relative error tolerance values (in this thesis denoted by AET and RET, respectively) and the maximum number of internal iterations allowed for a single integration step.

2.3.2 Stochastic simulation

A biochemical system should be modeled with stochastic approaches when some of its chemical species have a low concentration, so that the timing of reactions becomes random and the simulated trajectory diverges from the one predicted by a deterministic simulation. This allows the investigation of the emergent effects due to the intrinsic noise (e.g., bistability), allowing a deeper knowledge of the system's behavior. As a matter of fact, in a stochastic simulation only one of the possible trajectories actually occurs, according to a specific probability distribution. In particular, one may be interested into knowing the probability that the system will be in state $\mathbf{x}$ at time t given that it started in condition $\mathbf{x}_0$ at time t_0, that is, $P(\mathbf{x}, t|\mathbf{x}_0, t_0)$. This information can be obtained by calculating every possible outcome of the system, or by simulating multiple trajectories and analyzing the distribution of the states of the system. In both cases, the state of the system $\mathbf{x}$ is assumed to be discrete, i.e., it is a vector of integer-valued numbers corresponding to the exact amount of molecules for each chemical species.

Solving the CME

The traditional way to calculate the stochastic temporal evolution of a biochemical system Ω consists in solving the CME (Section 2.2.2), which describes the probability distribution function associated to Ω [339]. Unfortunately, the number of possible states of the biochemical system increases exponentially with the number of chemical species, since there is a specific differential equation for each possible state than can be reached as a consequence of reactions firing [145]. This characteristics leads to the so-called *curse of dimensionality* and prevents CME to be directly solved in practical applications. Nevertheless, numerical solution algorithms for the CME exist and they are usually based on matrix descriptions of the discrete-state Markov process [322]; anyway, because of the aforementioned problem of dimensionality, these methods are computationally expensive and not always feasible, especially for systems consisting of many molecular species, for which the number of reachable states is huge or even (countably) infinite.

Several analytical solution algorithms for the CME exist, for instance those based on uniformization methods [128, 312, 368], finite state projection algorithms [40, 222] or the sliding window method [357]; other methods were also introduced for special reaction systems characterized by particular initial conditions (see, e.g., [145] and references therein).

A different strategy to solve the CME consists in generating trajectories of the underlying Markov process, by means of stochastic simulation algorithms. They will be described in the next sections.

Gillespie's Stochastic Simulation Algorithm

A method for generating exact realizations of the CME is the Stochastic Simulation Algorithm (SSA), introduced by Gillespie in 1976 [100, 101], which provides trajectories of the associated continuous time, discrete state space jump Markov process $\mathbf{x}$ of a biochemical system Ω, whose initially conditioned density function is determined by the CME itself [102].

Briefly, starting from the system state $\mathbf{x}$, SSA determines which reaction will be executed during the next time interval $[t, t+\tau)$, by calculating the probability of each reaction R_j to occur in the next infinitesimal time step $[t+\tau, t+\tau+\mathrm{d}t)$, i.e., the joint probability $P(j, \tau|\mathbf{x}, t)$. Gillespie proved that this probability is proportional to $a_j(\mathbf{x})\mathrm{d}t$, being

$$a_j(\mathbf{x}) = c_j \cdot d_j(\mathbf{x}) \tag{2.4}$$

the propensity function of reaction R_j, where $d_j(\mathbf{x})$ is the number of distinct combinations of the reactant molecules occurring in R_j and c_j is a stochastic constant encompassing the physical and chemical properties of R_j [100]. The time τ before a reaction takes place is chosen according to the following equation:

$$\tau = \frac{1}{a_0(\mathbf{x})} \ln\left(\frac{1}{rnd_1}\right),$$

where rnd_1 is a random value sampled in $(0, 1)$ with a uniform probability, and $a_0(\mathbf{x}) = \sum_{j=1}^{M} a_j(\mathbf{x})$. The index j of the reaction to be executed is the smallest integer in the set $\{1, \ldots, M\}$ such that

$$\sum_{j'=1}^{j} a_{j'}(\mathbf{x}) > rnd_2 \cdot a_0(\mathbf{x}),$$

where rnd_2 is a second random value sampled in $(0, 1)$ with a uniform probability.

The algorithm described above is the traditional formulation of SSA, named *Direct Method* (DM). Gillespie also introduced a variant of this method, named *First Reaction Method* (FRM), which works by calculating a putative time τ_j for each reaction R_j and by applying the reaction having the smallest τ_j. Even though DM and FRM look conceptually different, they are provably equivalent [100].

In both algorithms, propensities are re-calculated after each simulation step, even when only a subset of propensities need to be actually updated. More specifically, only the propensities functions of reactions whose reactants were affected by a reaction fired in the previous simulation step must be updated. This is the rationale of the optimized SSA version proposed by Gibson and Bruck [99], based on FRM and called *Next Reaction Method* (NRM). NRM exploits a *dependency graph* which allows to update only the necessary propensity functions at each simulation step. The algorithm also improves the computational performances by reusing the putative times, avoiding the (computationally expensive) generation of random numbers. It also exploits optimized priority queues to store both propensities and putative times, so that updates are performed very efficiently.

Despite the improvement of performances provided by the NRM, SSA remains a computationally intensive algorithm, especially in the case of biochemical systems characterized by many reactions and chemical species. In [104], Gillespie introduced an approximate but faster version of SSA, called tau-leaping, designed to reduce the computational burden typical of SSA. SSA and tau-leaping share the characteristic that, even starting from the same initial state of the system, repeated executions of the algorithms will produce (usually quantitative, but potentially also qualitative) different temporal dynamics, thus reflecting the inherent noise of the system. These two algorithms, anyway, differ with respect to the way reactions are applied at each step: in SSA, only *one* reaction is applied, while with tau-leaping *several* reactions can be applied.

Tau-leaping

Given a state $\mathbf{x}$ of the system Ω, let $K_j(\tau, \mathbf{x}, t)$ denote the exact number of times that a reaction R_j would be fired in the time interval $[t, t+\tau)$; $\mathbf{K}(\tau, \mathbf{x}, t)$ denotes the probability distribution vector having $K_j(\tau, \mathbf{x}, t)$ as elements.

For arbitrary values of τ, the computation of the values $K_j(\tau, \mathbf{x}, t)$ can be as difficult as solving the corresponding CME. On the contrary, if τ is small enough so that the change in the state $\mathbf{x}$ 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 it is possible to evaluate a good approximation of $K_j(\tau, \mathbf{x}, t)$ by using the Poisson random variables with mean and variance $a_j(\mathbf{x})\tau$. So doing, the stochastic temporal evolution of the system is no longer exact (as in the case of SSA); however, the accuracy of tau-leaping can be fixed a priori by means of an error control parameter $\epsilon \in (0, 1]$, which is involved in the computation of the changes in the propensity functions and of the time increment τ.

The propensity functions change as a consequence of the modification in the molecular amounts of the reactant species, therefore the leap condition must be verified after each state update. This is achieved by evaluating an additional quantity $g_i = g_i(x_i(t))$ for each species S_i, which is related to the highest order $H(i)$ of the reactions in which S_i is involved as a reactant (see [46] for details). This information, along with the number of molecules of S_i involved in all highest-order reactions (given by the system state $\mathbf{x}$), is then used to bound the relative change of $x_i(t)$.

Starting from the state $\mathbf{x}$ and choosing a τ value that satisfies the leap condition, the state of the system at time $t+\tau$ is updated according to

$$\mathbf{x}(t+\tau) = \mathbf{x} + \sum_{j=1}^{M} \mathbf{MV}_j P_j(a_j(\mathbf{x}), \tau), \tag{2.5}$$

where $P_j(a_j(\mathbf{x}), \tau)$ denotes an independent sample of the Poisson random variable with mean and variance equal to $a_j(\mathbf{x})\tau$.

Note that the execution of many reactions per step could lead to negative amounts of the molecular species in Ω [104]. To be more precise, if the reactions executed during a step consume a number of reactant molecules greater than those occurring in the system, then negative species amounts would be generated; therefore, the simulation step cannot be executed. To avoid these situations, Cao *et al.* proposed a strategy that considers some reactions as *critical*: a reaction R_j is marked as critical if there are not sufficient reactant molecules to fire it at least θ_c times in the next time interval

[46]. A common threshold for critical reactions is $\theta_c = 10$, as suggested in [46]. At each iteration of tau-leaping, all reactions are partitioned into the sets of non-critical reactions (R_{nc}) and critical reactions (R_c). Only a single reaction belonging to R_c — selected following the SSA procedure — is allowed to fire during $[t, t+\tau)$.

The length of the step τ satisfying the leap condition is calculated as

$$\tau = \min_{i \in S_{nc}} \left\{ \frac{\max\{\epsilon x_i(t)/g_i, 1\}}{|\mu_i(\mathbf{x})|}, \frac{(\max\{\epsilon x_i(t)/g_i, 1\})^2}{\sigma_i^2(\mathbf{x})} \right\}, \tag{2.6}$$

where S_{nc} is the set of indices of reactant species not involved in critical reactions, and the values $\mu_i(\mathbf{x})$ and $\sigma_i^2(\mathbf{x})$ are computed as follows:

$$\mu_i(\mathbf{x}) = \sum_{j \in R_{nc}} \nu_{ji} a_j(\mathbf{x}), \quad \sigma_i^2(\mathbf{x}) = \sum_{j \in R_{nc}} \nu_{ji}^2 a_j(\mathbf{x}), \quad \text{for each } i = 1, \ldots, N. \tag{2.7}$$

If the execution of a tau-leaping step would lead to negative amounts of some species, then the τ value is halved and the number of reactions to execute is sampled *ex novo*. The problem of the negative species was also tackled by Tian and Burrage [332] and by Chatterjee *et al.* [55], using a different approach relying on binomial random variable to approximate $K_j(\tau, \mathbf{x}, t)$. Binomial random variables have two parameters: the expected value and the upper limit, here denoted by L_j. In both approaches, L_j values are chosen in such a way that $K_j(\tau, \mathbf{x}, t)$ is not allowed to consume more reactants molecules than are currently available, which would lead to negative values. Unfortunately, both methods are not robust and produce biased simulations, since multiple reactions can share common reactants and different reactions can have different stoichiometric values for the same reactant [45], so that the determination of L_j is far from trivial.

Finally, in Cao's tau-leaping algorithm [46], if τ is smaller than a multiple of $1/a_0(\mathbf{x})$ — which corresponds to the average time increment of SSA — then a certain number of SSA steps is executed because, given the actual state of the system, this will be more accurate and efficient than a tau-leaping step.

Stochastic differential equations

As described in Section 2.2.3, a biochemical system can be modeled by means of Chemical Langevin Equations, which are a special kind of SDEs. These equations cannot be solved (i.e., simulated) by means of common numeric integration algorithms, like those described in Section 2.3.1. Indeed, numeric solvers for SDEs must be employed, like the *Euler-Maruyama Method* [164, 318] which generalizes the EM by introducing a stochastic component.

Since $\mathcal{N}(\mu, \sigma^2) = \mu + \sigma\mathcal{N}(0,1)$, Equation 2.2.3 can be rewritten as:

$$\mathrm{d}\mathbf{x} = \sum_{j=1}^{M} \boldsymbol{\nu}_j a_j(\mathbf{x})\mathrm{d}t + \sum_{j=1}^{M} \boldsymbol{\nu}_j \sqrt{a_j(\mathbf{x})}\mathrm{d}W_j \tag{2.8}$$

which is the Itō form of the stochastic differential equation [318], where $W(t)$ is the stochastic component. More specifically, $W(t)$ is a *standard Wiener process* (also named *Brownian motion*), that is, a random variable continuously depending on $t \in [0, T]$ that satisfies the following three conditions:

1. $W(0) = 0$;
2. for $0 \leq s < t \leq T$, the random variable given by $W(t) - W(s)$ is normally distributed with $\mu = 0$ and $\sigma^2 = t - s$, i.e., with standard deviation equal to $\sqrt{t-s}$, so that $W(t) - W(s) = \sqrt{t-s} \cdot \mathcal{N}(0,1)$;
3. for $0 \leq s_1 < t_1 < s_2 < t_2 \leq T$ the increments $W(t_1) - W(s_1)$ and $W(t_2) - W(s_2)$ are independent.

Considering a discretized Brownian motion (i.e., a set of $\mathcal{D}$ values for t, at intervals $\Delta = \frac{T}{\mathcal{D}}$), from the three conditions follows that $W(t_j) = W(t_{j-1}) + \mathrm{d}W_j,\ j = 1, 2, \dots, \mathcal{D}$, where each $\mathrm{d}W_j$ denotes an independent random variable of the form $\sqrt{\Delta} \cdot \mathcal{N}(0,1)$. Hence, the single step of integration becomes

$$y_{n+1} = y_n + f(\mathbf{x})\Delta + g(\mathbf{x})\Delta W_n,$$

where f and g are the terms in the Itō form shown in Equation 2.8:

$$f(\mathbf{x}) = \sum_{j=1}^{M} \boldsymbol{\nu}_j a_j(\mathbf{x}), \qquad g(\mathbf{x}) = \sum_{j=1}^{M} \boldsymbol{\nu}_j \sqrt{a_j(\mathbf{x})}.$$

Gillespie pointed out that, when the ratio $\frac{a_j\tau}{\sqrt{a_j\tau}}$ is high, the noise term of Equation 2.8 can be neglected and the formula degenerates to the continuous-deterministic case of ODEs [103].

2.3.3 Multi-scale modeling and simulation

Theoretically, the simulation of mechanistic-stochastic models represents the most accurate way to investigate a biochemical system: all processes are explicitly considered, no approximations (e.g., mean field dynamics, reaction lumping) are made and reaction kinetics have a sound biophysical background [102], so that any emergent behavior due to complex interactions should emerge. Unfortunately, *exact* stochastic simulation algorithms have a huge computational cost, since they proceed by simulating a single reactions at a time.

The computational problem gets even worse when some reactions have a large stochastic constant or some of the chemical species are present in large amounts (e.g., metabolites). Both conditions have an impact on the propensity function by affecting the first and second term of Equation 2.4, respectively. Since the time step of the simulation τ is inversely proportional to $a_0(\mathbf{x})$, i.e., the sum of all propensity functions (see Equation 2.3.2), the simulation is slowed down by these *fast* reactions. Whilst this problem is partially mitigated by approximate methods like tau-leaping — in which multiple fast reactions are "compacted" into longer simulation jumps (see Equation 2.6) — for medium-large system this approach still suffers from the computation expense due to Poisson random numbers generation. Moreover, when a reaction involves a chemical species whose concentration is extremely high (e.g., ATP [226]), the role of noise can be neglected and a deterministic simulation would be as accurate as a stochastic simulation, allowing a huge computational speedup.

To the aim of simulating systems characterized by multiples scales (both in terms of reactions speed and molecular amounts), hybrid deterministic/stochastic modeling and simulation techniques have been introduced. These algorithms usually exploit a **partitioning** of the model into slow and fast reactions, or according to the molecular concentrations (see, for instance, [291]). Then, on the one hand, stochastic algorithms are employed to account for the stochasticity within the system to preserve the accuracy of the simulated dynamics concerning slow reactions and/or low molecular amounts. On the other hand, to the aim of speeding up the simulation, deterministic approaches are employed for the simulation of fast reactions and/or high molecular amounts (i.e., concerning the partition of the system in which the accurate description of the stochastic fluctuations of molecular species is not fundamental).

Existing hybrid algorithms generally partition the system and perform the simulation by using CLE and SSA, exploiting time-dependent probability densities, resulting in improved performances with respect to the tau-leaping algorithm. In the hybrid method proposed by Haseltine and Rawlings [125], a reaction belongs to the set of fast reactions if its propensity is greater than a propensity threshold, and the amount of all its reactants is greater than a population threshold. Fast reactions and slow reactions are simulated by means of CLE and SSA, respectively. Kiehl *et al.* [160] proposed a further optimization by partitioning the system and simulating the set of fast reactions by means of ODEs, reducing the computational effort due to the generation of normally distributed random deviates for high rate events. Salis *et al.* [292] proposed a modified approach in which the system is dynamically partitioned according to the characteristics of the reactions, i.e., whether they occur frequently in a small time interval, or they affect only small amounts of reactants or products. According to this strategy, it is possible to partition the system in such a way that the effects of slow reactions do not affect fast reactions, which can be simulated by means of ODE or SDE integrators. Harris and Clancy [122] proposed a "partitioned leaping" approach to tackle multi-scale modeling and simulation, which is based on a classification of reactions into four different partitions according to the propensities functions:

- $a_j\tau \lesssim 1 \rightarrow$ exact stochastic (SSA);
- $a_j\tau > 1$ but $\not\gg 1 \rightarrow$ Poisson (tau-leaping);
- $a_j\tau \gg 1$ but $\sqrt{a_j\tau} \not\gg 1 \rightarrow$ Langevin (SDE);
- $\sqrt{a_j\tau} \not\gg 1 \rightarrow$ deterministic (ODE).

It is worth noting that the dynamic partitioning of the system adds a relevant computational overhead to the simulation, potentially reducing the improvement of performance of the hybrid algorithm. Nevertheless, this approach avoids the need for domain knowledge and manual configuration of the simulator, which may be tedious for large size models. Moreover, dynamic partitioning allows to handle multistable and oscillatory systems, in which the amount of chemical species and the propensity of reactions vary in such a way that the initial partitioning is no longer convenient. Thresholds for reactions partitioning must be carefully selected, though, and no theoretical frameworks have been developed, yet.

Chapter 3

Evolutionary Computation and Swarm Intelligence

An optimization problem is the problem of finding the *optimal* solution in a N-dimensional *search space* of possible/feasible solutions (e.g., with respect to the a set of given constraints), exploiting mathematic and computational methodologies. Examples of optimization problems are minimization (or maximization) problems, whose goal is to identify the solution $\mathbf{x} \in \mathbb{R}^N$ that minimizes (or maximizes) a given *objective function* $\mathcal{F} : \mathbb{R}^N \to \mathbb{R}$. In what follows, the formalization of optimization problems will be related to the minimization goal, if not stated otherwise[1], that is, identifying the solution $\mathbf{x}$ such that $\mathcal{F}(\mathbf{x}) < \mathcal{F}(\mathbf{x}')$ for any $\mathbf{x}' \neq \mathbf{x}$.

In principle, a strategy for the exploration of the search space would be to enumerate all the feasible solutions, ranking them according to the value of the objective function. This is clearly an impracticable strategy because the search space may be too large or even infinite. For this reason, several algorithms have been proposed for a "smart" exploration of the search space.

3.1 Traditional optimization techniques

3.1.1 Simplex Method

The *Simplex Method* (SM) [67] allows to solve optimization problems belonging to the class of *Linear Programming*, i.e., problems in which a linear objective function is subject to linear (in)equality constraints. SM is an exact algorithm which represents

[1]It is worth noting that any maximization problem can be converted into a minimization problem by inverting the sign of the objective function.

the space of feasible solutions as a N-polytope, a convex space determined by the linear constraints (see Figure 3.1).

The SM works iteratively: given a initial vertex $\mathbf{x}^* \in \mathbb{R}^N$ of the N-polytope, it evaluates the objective function on all the vertexes that are connected to $\mathbf{x}^*$ by an edge. $\mathcal{A}$ denotes the set of all these vertexes. The best vertex in $\mathcal{A}$ with respect to the objective function (i.e., a vertex $\mathbf{x} \in \mathcal{A}$ such that $\mathcal{F}(\mathbf{x}) < \mathcal{F}(\mathbf{x}^*)$ and $\mathcal{F}(\mathbf{x}) < \mathcal{F}(\mathbf{x}')$, for any $\mathbf{x}' \in \mathcal{A}$, $\mathbf{x} \neq \mathbf{x}'$) is selected as the new $\mathbf{x}^*$ and the process is repeated. The algorithm stops when no adjacent vertexes improve the objective function (i.e., $\forall \mathbf{x} \in \mathcal{A} : \mathcal{F}(\mathbf{x}) \geq \mathcal{F}(\mathbf{x}^*)$): in such case, the last visited vertex $\mathbf{x}^*$ is guaranteed to be the optimal solution. One example of SM execution is depicted in Figure 3.1.

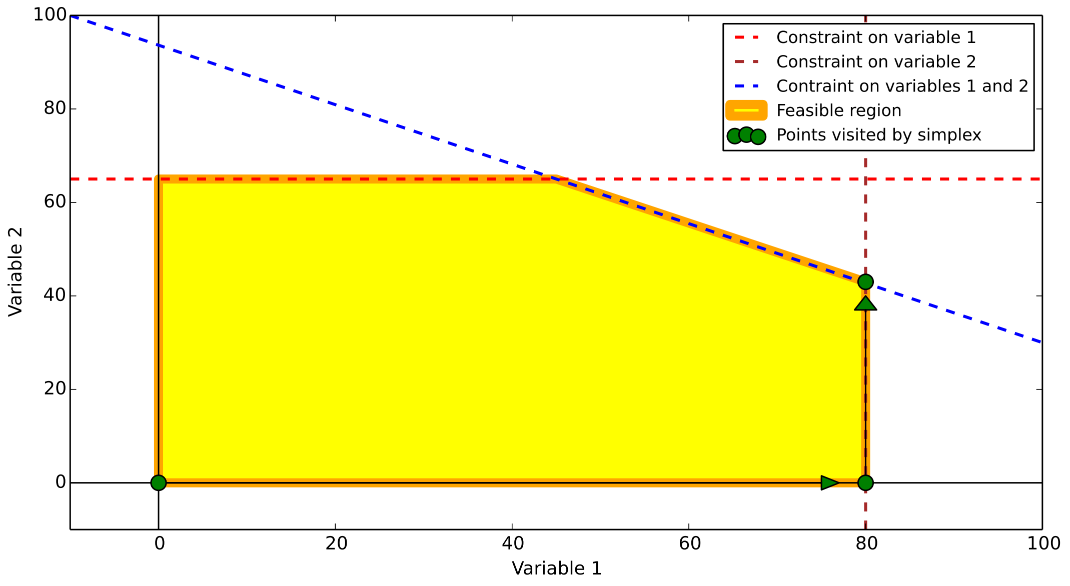


Figure 3.1: Example of execution of the Simplex Method. The 2-polytope defined by five constraints on the variables is represented by orange lines. The constraints are 5, consisting of 3 disequations (represented by the red, brown and blue dashed lines) and the non-negativity of the two variables. The feasible region defined by the constraints is represented by the yellow polygon. The algorithm starts by picking an initial vertex, that is, the initial candidate solution $\mathbf{x}^*$ of the problem. In the example, $\mathbf{x}^* = (0, 0)$. Then, by following the edges of the polytope, the SM moves across vertexes (green dots), according to the direction that improves the objective function (green arrows). The algorithm stops when no improving directions can be found: the last visited vertex is the global optimum.

Klee and Minty showed that, for some classes of optimization problems, the SM degenerates to an exponential worst case complexity [163]. Moreover, the standard SM requires a convex feasible region, linear constraints and a linear objective function. This represents a limitation, since many real case problems (e.g., the dynamical behavior of biological systems) are strongly non-linear. However, SM is still applied with success in Systems and Synthetic Biology, as it represents the foundation of *Flux Balance Analysis* [248]. In this methodology, the goal is to optimize the vector $\mathbf{x}$ of reaction fluxes in a biochemical system. In this case, the constraints are given by some biophysical limits (e.g., mass balance, energy balance, fluxes limitations) and the objective function generally consists in the maximization of some products of the system (e.g., biomass [38], ATP [272]). FBA can rely on SM thanks to a steady-state assumption on the concentration of chemical species, which allows to formalize the optimization of fluxes as a linear programming problem.

3.1.2 Gradient Descent

In the case of non-convex search spaces or non-linear objective functions, a different type of algorithm can be exploited: the *Gradient Descent* (GD) [92]. GD starts by choosing an initial guess $\mathbf{x}$ for the optimal solution and exploits the slope of the objective function. The initial solution $\mathbf{x}$ can be chosen either on a random basis, exploiting some *a priori* distribution, or using the available domain knowledge. Then, the gradient of the objective function is calculated in $\mathbf{x}$ and a new guess for the optimal solution is determined according to gradient's direction:

$$\mathbf{x} = \mathbf{x} - \Delta \cdot \nabla \mathcal{F}(\mathbf{x}) \tag{3.1}$$

where $\Delta \in \mathbb{R}^+$ is the *step size*. The algorithm stops when it reaches a stationary point (i.e., where $\nabla f(\mathbf{x}) = \mathbf{0}$), meaning that there is not an improving direction to follow. The Levenberg-Marquardt [199] algorithm adapts the step size Δ in Equation 3.1 during the optimization phase, so that the algorithm can converge to the optimal solution with increasing precision.

The main limitation of GD methods is that they converge to the global optimum only if the initial guess is chosen properly. Figure 3.2 shows a multi-modal objective function characterized by a local minimum $\mathbf{x}_1$, in position $(2, 2)$, and a global minimum $\mathbf{x}_2$, in position $(0, 0)$, which is the goal of the optimization. Since GD methods are strictly deterministic, if the initial guess is close to (2,2) there is no way for the algorithm to converge to $\mathbf{x}_2$. Because of this problem, GD is considered a *local optimization* methodology.

A first way to overcome this limitation is the *Multi-Start* strategy: a set of initial points is generated and they are used as starting points for the optimization. Even

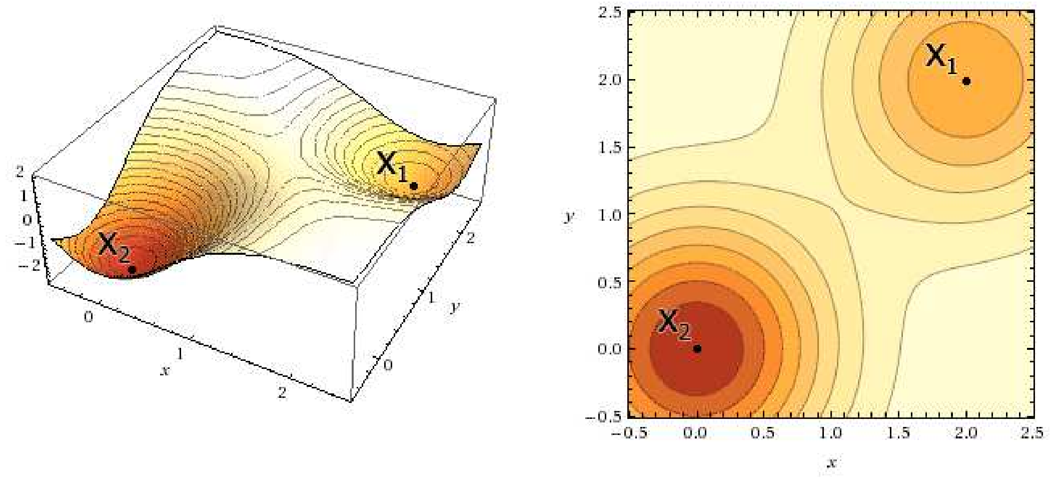


Figure 3.2: Example of an objective function with a local minimum $\mathbf{x}_1$ $(2, 2)$ and a global minimum $\mathbf{x}_2$ $(0, 0)$. A gradient descent method cannot converge to the optimum unless the initial guess is close to the basin of attraction of the global minimum $(0, 0)$.

though this methodology increases the probability of converging to the global optimum, it is not helpful in the case of large, noisy, multi-modal or high-dimensional search spaces, not to mention the case in which the objective function is not differentiable.

A second strategy is to introduce the stochasticity into the optimization process. One example is the *Simulated Annealing*, an optimization methodology that allows also the possibility of following a direction that *decreases* the quality of the candidate solution [161].

3.1.3 Simulated Annealing

Simulated Annealing (SA) is a probabilistic meta-heuristic, inspired by the annealing in metallurgy. It represents an adaptation of the Metropolis-Hastings algorithm [212], that is commonly used to generate sequences of random samples from complex multivariate distributions. The physical metaphor inspiring SA consists in the melting of metal and its successive slow cooling, to revert its crystalline form into a more ordered state. The current state of the system $\mathbf{x}$ can, in probabilistic terms, change to a neighbor state $\mathbf{x}'$ or remain in the same state, while the material cools down, looking for the lowest energy configuration. The concept of state corresponds here to a candidate solution of the optimization problem.

The neighborhood of a solution depends on the way solutions are structured: for instance, it could be a set of random points contained in a hypersphere centered in $\mathbf{x}$. Thus, in contrast to GD, SA does not require the function to be differentiable to explore the search space.

Each solution $\mathbf{x}$ has a different energy (i.e., a value of the objective function) denoted by $E(\mathbf{x})$. A probabilistic state switch leads the system to move towards states (i.e., solutions) of lower energy. The probability of accepting a new solution of higher energy is calculated using the following acceptance function:

$$P(\text{accept } \mathbf{x}' | E(\mathbf{x}), E(\mathbf{x}')) = \begin{cases} 1 & \text{if } E(\mathbf{x}) > E(\mathbf{x}'), \\ \exp(\frac{-|E(\mathbf{x})-E(\mathbf{x}')|}{T}) & \text{otherwise,} \end{cases} \tag{3.2}$$

where T is the current *temperature* of the system, which generally starts from 1 and tends to 0 during the iterations of the algorithm. Thanks to this strategy, SA avoids the entrapment into local minima and it was proven to asymptotically converge to the global optimum [161]. Hence, differently from the GD methods, it is considered a *global optimization* method.

A well known extension of SA is the *Tabu Search* [108]. In this algorithm, optimization is aided by a routine that marks the recently visited states, which are automatically excluded from the neighbors set of the current state.

In general, SA represents an improvement with respect to traditional local search techniques because it introduces the capability of *exploring* the search space, instead of just *exploiting* a promising neighborhood. More advanced optimization techniques are designed to have a good balancing between exploration and exploitation capabilities, that is generally controlled by some specific functioning settings, like the temperature of SA (the value T in Equation 3.2).

All the methodologies described so far exploit a single solution $\mathbf{x}$, which is iteratively improved until convergence. A completely different approach, based on a **population** of individuals, is the basis of Evolutionary Computation (EC) and Swarm Intelligence (SI) techniques, described in Sections 3.2 and 3.3 respectively.

3.2 Evolutionary Computation

EC exploits the Darwinian evolution theory to solve complex problems [69]. Many bio-inspired EC methods have been proposed (e.g., Genetic Algorithms (GA) [137], Evolution Strategy (ES) [29], Differential Evolution (DE) [324]), all sharing the following common traits:

- they exploit a population $\mathcal{P}$ of randomly generated individuals, i.e., the candidate solutions;

- $\mathcal{P}$ evolves, generation after generation, thanks to an iterative process that employs random modifications of the individuals;
- the individuals able to solve the problem better than the others, have a higher probability of being conserved and promoted during the evolutionary process;
- to discriminate the best solutions in $\mathcal{P}$, a *fitness function* quantifies the "quality" of each individual;
- the process is executed iteratively until a termination criterion is met (e.g., an optimal solution is found, the algorithm has performed a fixed number of generations or has reached a maximum execution time).

The fitness function has the same meaning of the objective function, it assesses the "goodness" of the individuals with respect to the optimization problem, and ultimately drives the evolution of the whole population. The hyper-surface described by the fitness function over the set of feasible solutions (see, e.g., Figure 3.2 or 6.2) takes the name of *fitness landscape*.

Another common feature of the existing EC methodologies is that they require the fine tuning of some functioning settings to obtain the best performances. A notable exception to this issue is represented by settings-free algorithms (e.g., parameter-free GAs [298], Tribes [61]), which trade convergence speed for easiness of use.

3.2.1 Genetic Algorithms

GAs were introduced by Holland in 1975 [137] as a global search methodology inspired by the mechanisms of natural selection. GAs exploit a population $\mathcal{P}$ composed of Q randomly created individuals that are usually defined as fixed-length strings over a finite alphabet, representing solutions of the problem under investigation. This characteristic makes GAs particularly suited for combinatorial optimization.

The individuals of the population undergo an iterative process whereby three genetic operators (selection, crossover, mutation) are applied, according to a given fitness function, to simulate the evolution process which results in a new population of possibly improved solutions.

During the selection process, individuals from $\mathcal{P}$ are chosen and inserted into a new temporary population $\mathcal{P}'$ using some fitness-dependent sampling procedure [19]. There exist multiple selection strategies, most notably:

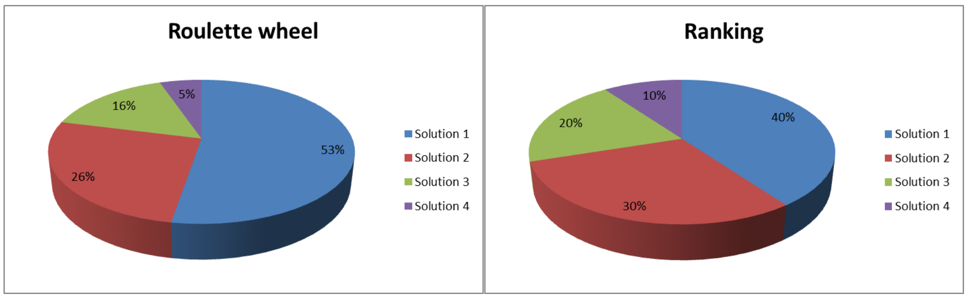
Roulette wheel The probability of selecting an individual is directly proportional to its fitness value. Denoting by $\mathcal{F}_i$ the fitness value of the i-th individual, the probability of the i-th individual of being selected and inserted in population $\mathcal{P}'$ is then equal to:

$$P(i \in \mathcal{P}') = \frac{\mathcal{F}_i}{\sum_{q=1}^{Q} \mathcal{F}_q}. \tag{3.3}$$

Ranking Individuals are ranked according to their fitness values, and the probability of selecting an individual is proportional to its *position* in the ranking. This methodology mitigates potential problems when the fitness values of candidate solutions differ too much, so that the individuals with worse fitness have a very low probability of being selected, causing a loss of diversity in the population. Unfortunately, this strategy can slow down the convergence if similar selection probabilities are assigned both to the best and worst individuals. A comparison of the probability distributions determined by the roulette wheel and the ranking selection mechanisms is shown in Example 3.2.1.

Tournament A subset of q individuals, $2 \leq q < Q$, is randomly chosen from $\mathcal{P}$. Then, the candidate solution having the best fitness is deterministically identified and copied into $\mathcal{P}'$. The value q is chosen by the user and represents the selection pressure that is applied to the population.

Example 3.2.1 *Comparison of the selection probability distributions determined by roulette wheel and ranking selection mechanisms. In this example, the individuals have the following fitness values: $\mathcal{F}_1 = 10$, $\mathcal{F}_2 = 5$, $\mathcal{F}_3 = 3$, $\mathcal{F}_4 = 1$. The selection probability calculated according to Equation 3.3 assigns to solution 4 a very low probability (figure on the left), which is doubled in the ranking mechanism (figure on the right).*



Once exactly Q individuals have been selected and inserted in population $\mathcal{P}'$, the crossover operator is used to combine the structure of two promising parents into new and improved offspring, which are collected into a third population $\mathcal{P}''$.

In what follows, $[i : j]_k$ denotes the substring from i-th to j-th symbol of a candidate solution k, while $\bullet$ denotes the concatenation operator between two strings. By assuming candidate solutions of length $\mathfrak{L}$, the most common crossover techniques between two parents $p_1, p_2 \in \mathcal{P}'$ can be described as follows:

One-point crossover A single index $1 \leq \chi_a < \mathfrak{L}$ in selected. The new offspring are created as $[1 : \chi_a]_{p_1} \bullet [\chi_a + 1 : \mathfrak{L}]_{p_2}$ and $[1 : \chi_a]_{p_2} \bullet [\chi_a + 1 : \mathfrak{L}]_{p_1}$;

Two-point crossover Two indexes $1 \leq \chi_a < \chi_b < \mathfrak{L}$ are selected. The new offspring are created as $[1 : \chi_a]_{p_1} \bullet [\chi_a + 1 : \chi_b]_{p_2} \bullet [\chi_b + 1 : \mathfrak{L}]_{p_1}$ and $[1 : \chi_a]_{p_2} \bullet [\chi_a + 1 : \chi_b]_{p_1} \bullet [\chi_b + 1 : \mathfrak{L}]_{p_2}$;

Uniform crossover Differently from one- and two-point crossovers, the uniform crossover does not exchange substrings but it exploits a *mixing ratio* which recombines smaller pieces of the parents.

A comparison of these three crossover techniques is provided in Example 3.2.2. The crossover between two individuals is performed with a probability p_c. If two individuals are not selected for crossover, they are copied identically into $\mathcal{P}''$.

Example 3.2.2 *Comparison of one-point, two-point and uniform crossover mechanisms. Assume the two following binary-valued GA individuals:*
$p_1 = [0, 1, 1, 0, 1]$, $p_2 = [1, 0, 0, 0, 0]$.
Considering one-point crossover, with $\chi = 3$*, the following substrings (denoted by bold characters) will be exchanged:* $p_1 = [\mathbf{0}, \mathbf{1}, \mathbf{1}, 0, 1]$, $p_2 = [\mathbf{1}, \mathbf{0}, \mathbf{0}, 0, 0]$. *This yields the following offspring individuals:* $p_1' = [1, 0, 0, 0, 1]$, $p_2' = [0, 1, 1, 0, 0]$.
In the case of two-point crossover, with $\chi_a = 2$ *and* $\chi_b = 4$*, the following substrings (denoted by bold characters) will be exchanged:* $p_1 = [\mathbf{0}, \mathbf{1}, 1, \mathbf{0}, \mathbf{1}]$, $p_2 = [\mathbf{1}, \mathbf{0}, 0, \mathbf{0}, \mathbf{0}]$. *This yields the following offspring individuals:* $p_1' = [1, 0, 1, 0, 0]$, $p_2' = [0, 1, 0, 0, 1]$.
Finally, assuming that a uniform crossover with a mixing ratio equal to $2/5$ *selects the indexes* 1 *and* 5 *(denoted in bold), it means that the symbols that will be swapped are* $p_1 = [\mathbf{0}, 1, 1, 0, \mathbf{1}]$, $p_2 = [\mathbf{1}, 0, 0, 0, \mathbf{0}]$. *This yields the following offspring individuals:* $p_1' = [1, 1, 1, 0, 0]$, $p_2' = [0, 0, 0, 0, 1]$.

Finally, the mutation operator is used to perturb the encoding of individuals in $\mathcal{P}''$, allowing a further exploration of the search space. Mutation alters a symbol of the individual, which is substituted by a random symbol from the alphabet, with a fixed probability p_m.

After the application of genetic operators, individuals in $\mathcal{P}''$ replace those in $\mathcal{P}$ and the process iterates until a halting criterion is met, e.g., after a fixed number of generations.

GAs can be extended to support the optimization of real-valued problems: Real-Coded GAs (RCGAs) [131] exploit a different representation of individuals based on strings of floating point numbers, instead of symbols from a finite alphabet. Of course, both mutation and crossover operators are adapted to the new case. The former is usually based on a random perturbation of the values; the latter is more complicated, since it is generally implemented as a weighted interpolation of the parents. A review of the crossover operators for RCGAs is available in Herrera *et al.* [130].

GAs are characterized by a well-known convergence theorem named *schema theorem*, proved by Holland [137]. The theorem ensures that the presence in $\mathcal{P}$ of a schema H (a template of solutions), having a good impact on the fitness value, increases exponentially generation after generation.

3.2.2 Evolution Strategy

ES is an EC technique introduced during the 60s and further developed by I. Rechemberg [274] for the optimization of problems in continuous search spaces. In ES, the Q individuals of the population $\mathcal{P}$ represent positions in the $\mathbb{R}^N$ domain. Traditionally, in ES the population size is denoted by μ. Differently from GAs, ES exploits a selection procedure at the end of the recombination phase, according to the following mechanism:

- a set of individuals from population $\mathcal{P}$ is picked up and new offspring are generated by duplication and/or recombination of the parents;
- the new offspring undergo a mutation phase and are inserted in population $\mathcal{P}$;
- a mechanism of selection reduces the population size down to μ.

In ES, the population size is traditionally denoted by μ (i.e., $\mu = Q$). The number of individuals exploited for recombination is denoted by ρ, with $2 \leq \rho \leq \mu$. This is different from GAs, since the latter generally exploit *exactly* two parents for the crossover mechanism [81]. The number of new offspring created during a generation is denoted by λ (generally, $\mu \leq \lambda$).

The recombination of the ρ parents can be based on a discrete combination of the components of the real-numbered vectors encoding the individuals, or on a weighted average of the ρ parents. Since recombination is not exploited in this thesis, it will not be described any further. A more detailed explanation can be found in Hansen *et al.* [118].

The mutation operator is generally implemented as the perturbation of the coordinates of the individual. This perturbation is generally drawn from a multivariate normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{C})$ with zero mean and covariance matrix $\mathbf{C} \in \mathbb{R}^{N \times N}$ [304]. The covariance matrix $\mathbf{C}$ can be controlled to generate isotropic, axis-aligned or completely general perturbations, and can evolve with the solutions in order to possibly exploit any correlation between the components in solutions. Figure 3.3 shows these three multivariate distributions. The self-adaptation of the covariance matrix to the

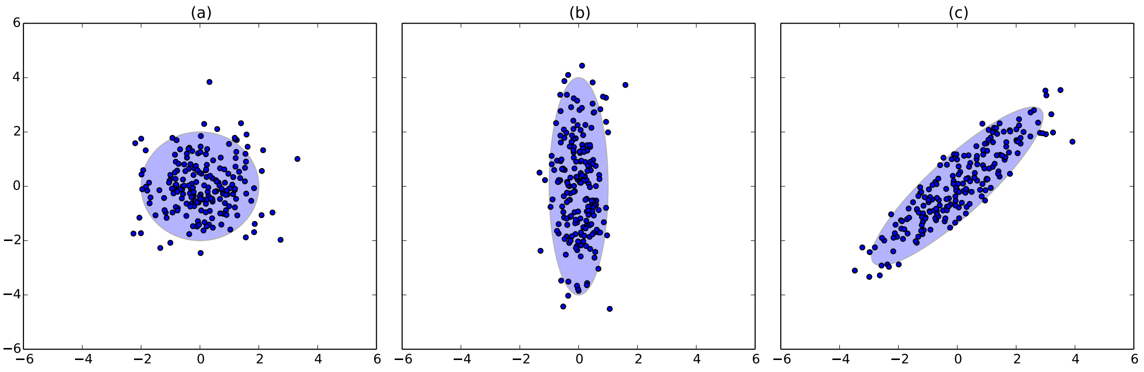


Figure 3.3: Three examples of multivariate bi-dimensional normal distributions ($\mathcal{N}(\mathbf{0}, \mathbf{C}_1)$,$\mathcal{N}(\mathbf{0}, \mathbf{C}_2)$ and $\mathcal{N}(\mathbf{0}, \mathbf{C}_3)$), used for individuals mutation in ES. (a) Isotropic ($\mathbf{C_1} = \mathbf{I}$); (b) axis-aligned ($\mathbf{C}_2$ is a diagonal matrix); (c) completely general ($\mathbf{C}_3$ has the same eigenvalues as $\mathbf{C}_2$, but it results in a differently rotated ellipsoid). The distribution produced by $\mathbf{C}_3$ allows to exploit possible correlations between the components of the solutions.

problem under investigation is the foundation of *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES) [119], considered among the state-of-the-art optimization algorithms.

The selection operator traditionally applies a *truncation selection* in which the best μ individuals — with respect to their fitness values — survive in the next generation. ES can exploit two specific methods for the selection and creation of a new population, namely *plus* (+) and *comma* (,) selections. In plus-selection, the best μ of $\mu + \lambda$ individuals are chosen. In comma-selection, all parents die and the best μ of λ individuals survive to the next generation. The $(1 + \lambda)$-ES is a specific case of plus-selection in which only the best individual (including the parents) survives, generation after generation, and it is exploited to generate new offspring. The $(1 + \lambda)$

ES is exploited in this thesis for the evolution on candidate biochemical networks in Chapters 7 and 8.

A global convergence theorem for any version of ES, similar to Holland's schema theorem, is still missing. Since SA can be seen as a $(1+1)$-ES with a time-dependent selection pressure [30] with a fixed mutation strength, a proof of convergence based on the same concept was proposed by Born [32]. For the case of $(\mu+\lambda)$-ES, no theorem has been presented yet, while it seems unlikely to prove the convergence of (μ,λ)-ES, since all parents are discarded after each generation.

3.2.3 Genetic Programming

GAs are particularly efficient in solving a specific instance of a complex problem, but they are not suitable for solving *classes* of problems. One way to encompass whole classes of problems is to embed variables in the encoding of the candidate solutions. Genetic Programming (GP), introduced by Koza in 1992, is an EC technique similar to GAs except that it evolves populations of *computer programs* [168], which can obviously contain variables.

Since the syntax of traditional programming languages is extremely complex, the definition of genetic operators (e.g., crossover) for the GP could be a complex task. For this reason, individuals are generally represented by recursive algebraic structures like derivation trees or LISP S-expressions. In the former case, the search space consists in all the trees that can be constructed using elements from two sets:

- a set of functional nodes (inner nodes);
- a set of terminal nodes (the leaves, consisting in constants and variables).

The proper choice of these two sets is vital for the correct functioning of GP, because they must satisfy two important conditions: *closure* and *sufficiency*.

Closure property

It is satisfied when derivation trees are always well defined and consistent, also after the execution of a genetic operator. In particular, functions implemented in functional nodes must satisfy two sub-properties: *type consistency* and *evaluation safety* [262].

- Type consistency depends on the fact that crossover can mix arbitrary subtrees, so that functions must be able to handle *any* kind of subtree as an argument. All type conversions that may be needed must be considered and implemented. For instance, an IF statement expects a boolean predicate as first argument, but an object of a different type (e.g., a floating point number, a string) could instead be present because of the evolutionary process: any unexpected value must be correctly converted into the expected type.

- Evaluation safety means that the execution of functions must be robust with respect to run-time failures and exceptions (e.g., a logarithm of a negative number, a division by zero). This problem can be mitigated by using *protected* versions of the operators, which test the input before actually performing the computation. Another approach consists in considering a penalty score in the fitness function for those individuals throwing exceptions during the execution.

Sufficiency

It states that nodes in the functional and terminal sets must be sufficient to build the optimal tree. Since the optimal tree is also the goal of the optimization, which is unknown in advance, the property of sufficiency is far from straightforward to be achieved. Nevertheless, if the functional set is insufficient, GP can only converge to approximate solutions instead of the optimal program.

Genetic operators in GP are similar to those described for GAs: selection, crossover, and mutation. Selection, in particular, is identical to GAs. Crossover is defined as the exchange of random subtrees between two individuals. Mutation consists in replacing a random node with a randomly generated tree.

Differently from GAs, whose individuals generally have a fixed size [109], GP's genetic operators can change the size of candidate solutions (e.g., the height of the derivation trees or, equivalently, the depth of nested S-expressions). For this reason, GP is prone to the issue of *bloating*, that is, the uncontrolled growth of individuals, generation after generation. Bloating should be avoided, because smaller programs have better generalization capabilities and require less time to run [142, 367]. The issue of bloating is particularly delicate, since the structure of the optimal solution is unknown and any *a priori* limitation of the maximum height of individuals may prevent the derivation of the optimal tree. On the other hand, a penalization term in the fitness function, proportional to the complexity of the tree, may push the GP towards local optima instead of converging to the optimal solution.

A schema theorem, equivalent to the GA counterpart and proving the asymptotic convergence of GP, was presented by Poli and Langdon in 1998 [261].

3.2.4 Cartesian Genetic Programming

Cartesian Genetic Programming (CGP) is an EC method based on GP whose individuals are described by means of indexed graphs, rather than derivation trees, having (sequentially numbered) nodes arranged in a Cartesian coordinates grid [214]. The genotype of each individual is therefore represented by a sequence of node connections and functions in the grid, and can be formally mapped onto a fixed-length vector of integers, which is called *Cartesian Program* (CP).

Formally, a CP is a 9-tuple $\{G, n_i, n_o, \mathfrak{F}, FN, n_r, n_c, n_n, l\}$ where:

- G is the genotype, encoded as a vector of integer numbers that represent the connections from the input nodes to the output nodes of the grid;
- $n_i, n_o \in \mathbb{N}$ are the number of input and output nodes, respectively;
- $\mathfrak{F}$ is a finite set of functions (for instance, elementary arithmetic operations as $\{+, -, *, /\}$);
- FN is a grid of functional nodes, sequentially indexed by rows and columns, each one containing a function from F;
- $n_r, n_c \in \mathbb{N}$ are the number of functional nodes appearing in each row and in each column of the grid, respectively, so that $|FN| = n_r n_c$;
- $n_n \in \mathbb{N}$ is the number of input connections in each functional node;
- $l \in \mathbb{N}$ is the so-called *"levels back"* parameter, a measure of the CP inter-connectivity which determines how many preceding columns (in the grid of functional nodes) can have their output connected to the functional nodes appearing in any given column of the grid.

The connections of a CP start from the input nodes and pass through the functional nodes, each one having a fixed number n_n of input connections. The length of the genotype G is equal to $n_r n_c(n_n + 1) + n_o$, that is, an integer number is assigned to each input connection and to the output connection of every functional node, as well as to each output node of the grid. The inter-connectivity of nodes generally exploited in CGP, and specifically considered in this thesis, is strictly feed-forward, meaning that nodes belonging to the same column of the grid cannot be connected to each other. In addition, any node can be either connected or disconnected; disconnected nodes represent non-coding genes in the genotype and are ignored in the phenotype. The phenotype of a given CP is the actual graph that the CP represents. In this thesis, the semantics of the phenotype corresponds to a set of biochemical reactions derived from the graph, that will evolve in the Reverse Engineering and Evolutionary Design processes described in Chapters 7 and 8, respectively.

In a CGP population, the fitness function of each CP is evaluated and the best candidate solutions are selected to generate the offspring by means of a mutation operator. During the iterative process, CGP evolves a population of individuals characterized by a set of expressions that are formed by the composition of input nodes and functional nodes. The total number of possible expressions represented by a given CP is upper bounded by the number n_o of output nodes. If an output node is linked to a disconnected functional node, some internal nodes will not be part of any path connecting an input node to that output node, a circumstance that leads to the possible existence of different genotypes mapping to the same phenotype. These non-coding regions are however important in CGP for three reasons:

1. they reduce the size of the phenotype, as stated above;

2. they reduce the effect of mutations, since a mutation acting on a disconnected functional node will not contribute to the variation of the phenotype;

3. even if they are unused in the current CP, a mutation could suddenly connect some disconnected node to the rest of the graph, thus resulting in a relevant change in the phenotype.

To the best of my knowledge, at the moment of writing no schema nor convergence theorems for CGP have been proposed. Nevertheless, this EC method has been applied to a large number of problems in multiple disciplines (e.g., medicine [15], economy [363], image processing [120]).

In this thesis, candidate solutions evolve by using a (1+λ)-ES [29], as described in [214]: all the individuals are evaluated and the best one is selected as a parent for the next generation. Then, λ offspring are produced by means of random mutations, that is, random modifications of the integers which constitute the genotype of the parent individual. The proportion of genes that are mutated is determined by the mutation rate parameter $\rho \in (0, 1)$. The ES methodology does not exploit any crossover mechanism.

In Chapter 7 it is shown a methodology to exploit CGP for the reverse engineering of biochemical reaction models; Chapter 8 contains the definition of a method to evolve novel biochemical reaction models, modeling gene regulation circuits.

3.3 Swarm Intelligence

Differently from EC techniques, SI takes its inspiration from the emergent collective behavior of groups of living organisms. According to sociobiological investigations, some animals and social insects, like those belonging to the Hymenoptera order (e.g., ants, bees) [138], have behavioral patterns that, collectively, allow groups to self-organize, share information and perform complex tasks that a single individual would not be able to carry out [305]. SI techniques are inspired by such behaviors, and are exploited to design nature-inspired holistic optimization techniques.

In the next sections, a brief review of SI methods and a detailed description of Particle Swarm Optimization, arguably the most known SI technique, are provided.

3.3.1 Hymenoptera-based SI techniques

One of the most exploited SI techniques is *Ant Colony Optimization* (ACO) [77], a method based on the mechanism of *stigmergy*, the indirect communication between agents typical of pheromone-based ants signaling [331]. Foraging ants, indeed, tend to deposit a pheromone trail along a route leading to food. When other ants meet the pheromone trail, they tend to follow that route to reach food, reinforcing the signal. Eventually, the optimal route emerges from the collective movement of the colony, while sub-optimal trails slowly evaporate. In ACO, simulated pheromone trails are used to stochastically generate and iteratively improve a set of candidate solutions. Convergence theorems were proposed for this powerful combinatorial optimization algorithm [116] and was proven to be effective in tackling problems belonging to the NP complexity class, like the traveling salesman (TSP) [325] or the maximum independent set [183].

Like ACO, also *Artificial Bee Colony* (ABC) exploits the emergent behavior of a population of virtual insects, taking inspiration from the collective behavior of foraging honey bees. This optimization method exploits virtual worker, onlooker and scout bees [150] which cooperate in identifying the best food resources (i.e., solutions with the best fitness). In particular, scouts are responsible for the exploration of the search space and become workers when they identify a promising food source (i.e., a region with good fitness). During the iterative process, onlookers are subdivided into groups which randomly exploit every food source assigned to a worker bee. ABC is a global optimization method, designed to explore real-valued search spaces, which was shown to be competitive with respect to other SI and EC techniques [149]. Combinatorial modifications were applied to the TSP [151] and other problems of the NP class, even exploiting the memetic approaches [270] that will be described in Section 3.4.

3.3.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a swarm-intelligence population-based optimization meta-heuristic, inspired by the social behavior of bird flocking or fish schooling, introduced by Kennedy *et al.* [157]. In PSO, a population (the *swarm*) of n candidate solutions (the *particles*) moves in a M-dimensional search space and cooperates to identify an optimal solution. Each particle i ($i = 1, \dots, n$), is characterized by a position $\boldsymbol{\gamma}_i \in \mathbb{R}^M$ and by a velocity $\mathbf{v}_i \in \mathbb{R}^M$, that is used to update its position.

The PSO concept consists in changing, at each iteration, the velocity of each particle towards some attractor, typically its best position $\mathbf{b}_i \in \mathbb{R}^M$ found so far, and the global best position $\mathbf{g} \in \mathbb{R}^M$ found by the swarm. The update procedure continues until some termination criterion is met; in this thesis, the optimization process is halted when a maximum number of iterations $IT_{\texttt{MAX}}$ is reached.

The behavior of the swarm is influenced by two parameters: the social attraction $C_{soc} \in \mathbb{R}^+$ and the cognitive attraction $C_{cog} \in \mathbb{R}^+$. These parameters control the global exploration and local exploitation of the search space, and they are weighted by two vectors $\mathbf{r}_1, \mathbf{r}_2 \in \mathbb{R}^M$ of random numbers sampled with uniform probability in [0,1] to prevent particles from prematurely converging to local minima. The velocity of particles is also limited to a maximum magnitude $v_{\texttt{MAX}} \in R^+$, and weighted by an inertia factor $w \in \mathbb{R}^+$ to avoid chaotic behaviors of the swarm. This leads to the following definition of the velocity update function for a generic i-th particle:

$$\mathbf{v}_i = w \cdot \mathbf{v}_i + C_{soc} \cdot \mathbf{r}_1 \circ (\boldsymbol{\gamma}_i - \mathbf{g}) + C_{cog} \cdot \mathbf{r}_2 \circ (\boldsymbol{\gamma}_i - \mathbf{b}_i), \tag{3.4}$$

where $\circ$ denotes the component-wise multiplication operator. Then, the positions of the particle is updated by calculating

$$\mathbf{x}_i = \mathbf{x}_i + \mathbf{v}_i \tag{3.5}$$

for all $i = 1, \dots, n$.

The value of w may be kept constant throughout the optimization process, or change according to some update function. Several works analyzed the performances of the PSO with different settings, since they have a relevant impact on the optimization performances. For instance, some works focused on the optimal choice of parameters C_{soc}, C_{cog} [17, 42, 337] and w [54]. These settings might also self-adapt during the

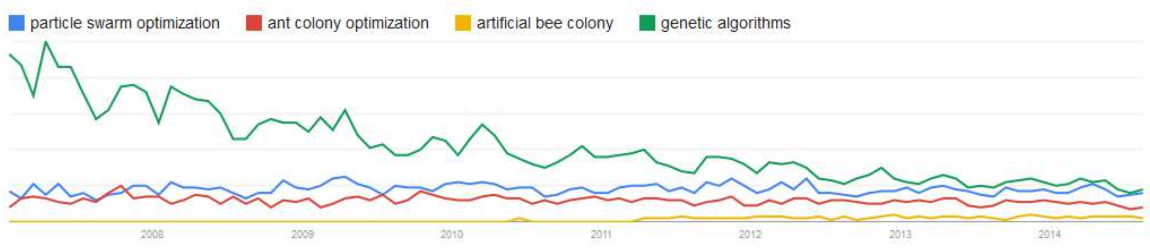


Figure 3.4: Trend of search terms about global optimization used on Google, between 2007 and 2014. The most known Swarm Intelligence methods (PSO, ACO, ABC) are compared to GAs, which are among the most exploited global search methods. The result shows that interest about GAs decreases, while about SI methods increases. In particular, PSO is well established as the most attractive SI method, while researchers started considering ABC only in the recent years.

optimization phase, for instance by exploiting fuzzy rules [1, 309]. The choice of the best functioning settings is generally driven by the problem under investigation and, in particular, by the characteristics of the fitness landscape. In Section 6.2.2, an analysis of the influence on the parameter estimation problem of some PSO settings is presented and discussed.

In PSO, the search space is bounded to prevent particles from diverging towards infinity: as soon as a particle moves outside the search space, some strategy is applied to keep it inside the valid region, for instance by placing the particle back on the space boundary or by making it change its direction. Examples of boundary conditions are the *absorbing*, *reflecting* and *damping* strategies [361], which relocate the particles inside the allowable solution space, so that the fitness function can always be evaluated, obtaining valid solutions to the optimization problem. In the case of absorbing condition, when a particle exits the allowable solution space along one dimension, it is relocated at the boundary of the solution space in that dimension and the relative velocity component is set to zero. The reflection condition is similar to the absorbing, but the velocity is reversed instead of being set to zero. Finally, in the damping condition, the velocity is reversed as in the reflecting condition, but it is also modulated by a random value sampled from $[0, 1]$ with uniform distribution. In this work, β_j^{min} and β_j^{max} denote the lower and higher boundaries on the j-th dimension of the search space, respectively.

Despite the lack of a proper convergence theorem, PSO was successfully applied to a plethora of problems [260] in many different disciplines. Its implementation simplicity and impressive optimization performances make it an attractive and popular methodology to solve complex problems (Figure 3.4).

3.4 Memetic approaches and open issues

Local search techniques like GD are very efficient in the case of unimodal fitness landscapes, in terms of performances and quality of solutions. As described in Section 3.2, for more complex problems, they get easily stuck in local minima, so that only a multi-start methodology that exploits a large number of starting points can increase the probability of converging to the global optimum [253].

On the other hand, stochastic global optimization algorithms (e.g., PSO) might be unable to perfectly exploit the neighborhood of a promising solution, because the structure of an individual might be perturbed in a direction that does not lead to a better region of the search space.

For this reason, in the latter years, part of the research about optimization algorithms focused on the hybridization of global and local search strategies. In particular, the so-called *Memetic Algorithms* (MAs) rely on global optimization methods embedding local search improvements. The idea is to combine the exploration capabilities of EC with the fine exploitation of neighborhoods performed by local search [169].

Despite this improvement, according to the *no free lunch* (NFL) theorem [359], there is no such a thing like the "perfect" optimization method. It can be proven that any couple of optimization algorithms a_1 and a_2 are equivalent when their performances are averaged across *all* possible problems. Formally, if $P(d_m)$ denotes the probability of observing a specific fitness d_m in m iterations then

$$\sum_{o \in \mathcal{O}} P(d_m | o, m, a_1) = \sum_{o \in \mathcal{O}} P(d_m | o, m, a_2)$$

where $\mathcal{O}$ is the space of all possible optimization problems. From the NFL theorem follows that each optimization algorithm performs better than the others on a specific subset of problems, while it performs worse (even than a random search) on the rest of the problems. The difficulty therefore consists in the selection of the proper algorithm for the specific problem under investigation.

Another difficulty of population-based approaches is the underlying computational complexity. As a matter of fact, hard optimization problems generally require a very expensive fitness evaluation, which is generally more computationally burdensome than the EC technique itself. The problem is further complicated by the fact that the computational complexity is proportional to the size of the population, which can be extremely large. Nevertheless, since all individuals are independent, the fitness evaluation can be straightforwardly accelerated by means of a parallel architecture. In the context of the problems tackled in this thesis, there are some notable examples of large parallel infrastructures used for this purpose. For instance, the cluster exploited by Koza *et al.* for the reverse engineering of a biochemical system (see Chapter 7) was equipped with 1000 processing nodes, used to distribute the fitness evaluations of 100000 GP individuals [167].

Among the existing parallel architectures, there exists a power-efficient, performant and cheap alternative: the General-Purpose GPU computing, subject of the next chapter.

Chapter 4

General-purpose GPU computing

The emerging field of *General-Purpose GPU* (GPGPU) computing allows developers to exploit the great computational power of modern multi-core *Graphics Processing Units* (GPU), by giving access to the underlying parallel architecture that was conceived for speeding up real-time three-dimensional computer graphics [227]. Modern GPUs are, indeed, multi-core coprocessors designed to process a massive number of geometric primitives, mainly triangles (ordered triples of vertexes), characterized by multiple attributes, in a multi-threaded and pipelined fashion.

Initially designed to have a fixed functionality for vertex transform and lighting (T&L) — which could be exploited by means of libraries like OpenGL and Microsoft Direct3D — GPUs were later equipped with programmable vertex and fragment processors, following the requests of video-games developers and 3D artists. A program for custom T&L takes the name of *shader*, following the naming convention introduced by Pixar in 1988 [259]. New languages were proposed to simplify the development of shaders, since the introduction in 2001 of the Nvidia GeForce 3, the first programmable GPU. Nvidia defined its own shading language named CG, inspired by the C language, in 2003 [198]. The same year, Microsoft added HLSL (High-Level Shading Language) to the DirectX libraries [112]. The Khronos Group consortium, which controls the specification of the OpenGL API, proposed in 2004 the shading language GLSL (OpenGL Shading Language).

It was only a matter of time before the purpose of shaders changed into general computation instead of graphics-oriented geometric transformations. This approach, which gave birth to the GPGPU computing, was initially tricky since the input and the output of the computation were graphics objects (3D meshes, textures, pixels) and the computation itself was performed using a graphics API. The problem was solved with the introduction of APIs for the general-purpose computation, most notably

CUDA (created by Nvidia), OpenCL (proposed by the Khronos Group) and Microsoft DirectCompute.

Thanks to these new libraries, GPGPU computing nowadays represents an alternative to the traditional high-performance computing infrastructures (e.g., clusters of machines), characterized by low-costs and a reduced energy consumption, allowing the access to the tera-scale computing on common workstations of mid-range price. Nevertheless, a direct porting of sequential code on the GPU is most of the times unfeasible, due to the innovative architecture and the intrinsic limitations of this technology. As a consequence, the full exploitation of GPU's computational power and massive parallelism is challenging [89].

4.1 Nvidia CUDA

In order to simplify the development of GPU-bound general-purpose programs, Nvidia introduced in 2006 the GeForce 8800, the first unified GPU architecture for both graphics and computing [188]. For this GPU, Nvidia developed a new parallel computing platform and programming model named CUDA (Compute Unified Device Architecture) fully programmable in C, characterized by new multi-core *Streaming Multiprocessors* (SMXs) able to indifferently execute computing threads along with vertex, fragment, geometry and pixel shaders. CUDA is a mature architecture, providing developers with several additional libraries which make GPGPU computing easier. One of these libraries, CURAND, is described in details in Section 4.2.

The CUDA library is cross-platform and its programming model combines the *Single Instruction Multiple Data* (SIMD) architecture with multi-threading. CUDA automatically handles the control flow divergence, that is, threads can take different execution paths in a transparent way for the programmer. Nevertheless, because of the underlying architecture, conditional branches should be avoided as much as possible since they cause a reduction of performances, due to the serialization of the execution until re-convergence. For this reason, existing algorithms require a major redesign in order to reduce the need for conditional branches.

Following the naming conventions used in CUDA, a C/C++ function, called *kernel*, is loaded from the host (the CPU) to the devices (one or more GPUs) and replicated in many copies named *threads*. Threads can be organized in three-dimensional structures named *blocks* which, in turn, are contained in three-dimensional *grids* (as schematized in Figure 4.1, left side). Whenever the host computer runs a kernel function, the GPU creates the corresponding grid and automatically schedules each block of threads on an available SMX, allowing a transparent scaling of performances on different devices (see Figure 4.2).

In general, different devices are characterized by different architectures. Each architecture, named after a famous physicist, supports up to a specific CUDA version (with complete retro-compatibility), giving access to specific functionalities (named *compute capabilities*) (CC) and imposing different limitations. An brief overview of the compute capabilities and their corresponding functionalities is reported in Table 4.1. For the sake of brevity, the table summarizes only the novelties that are relevant for the present thesis. Additional information can be found in the latest (at the time of writing) CUDA programming guide [247].

Table 4.1: Overview of the new functionalities introduced by different CUDA compute capabilities and architectures.

Compute capability	*CUDA architecture*	*Functionalities introduced by the compute capability*
1.0	Tesla	2D grid blocks, up to 2^9 threads per block and $2^{16}-1$ blocks per grid, up to 16 KB of shared memory per SMX, up to 8 simultaneous blocks per SMX, up to 63 registers per thread.
1.1		Atomic functions in global memory.
1.2		Atomic functions in shared memory and warp voting.
1.3		Double precision floating point numbers.
2.0	Fermi	3D grid blocks, up to 2^{10} threads per block, up to 48 KB of shared memory per SMX, L2 cache on global memory, custom balancing of L1 cache and shared memory (48+16 KB vs 16+48 KB).
2.1		Advanced synchronization functions.
3.0	Kepler	Unified memory programming, up to $2^{31}-1$ blocks per grid, up to 16 simultaneous blocks per SMX, up to 255 registers per thread, balanced L1 cache/shared memory configuration (32+32 KB).
3.5		Dynamic parallelism.
5.0	Maxwell	Up to 64 KB of shared memory per SMX, up to 32 blocks per SMX.
5.2		Up to 96 KB of shared memory per SMX.

Table 4.1 shows how CUDA poses limitations on the number of threads a block may contain: since the introduction of the Fermi architecture, up to 1024 threads can be distributed in the three dimensions, given that x- and y-dimension must not exceed 1024 threads, while z-dimension cannot exceed 64 threads. However, the number of simultaneous threads is further limited by the available resources on the SMX (e.g,

registers, high performance memories). In standard CUDA, each block contains the same number of threads; however, this limitation was removed since CC 3.5, with the introduction of *Dynamic Parallelism* which allows threads to launch new grids [243].

When a SMX is given one or more blocks to execute, it partitions them into groups of 32 threads named *warps* which execute one common instruction at a time. The full efficiency is achieved when threads belonging to the same warp agree on their execution path, and when the number of threads can be divided by 32.

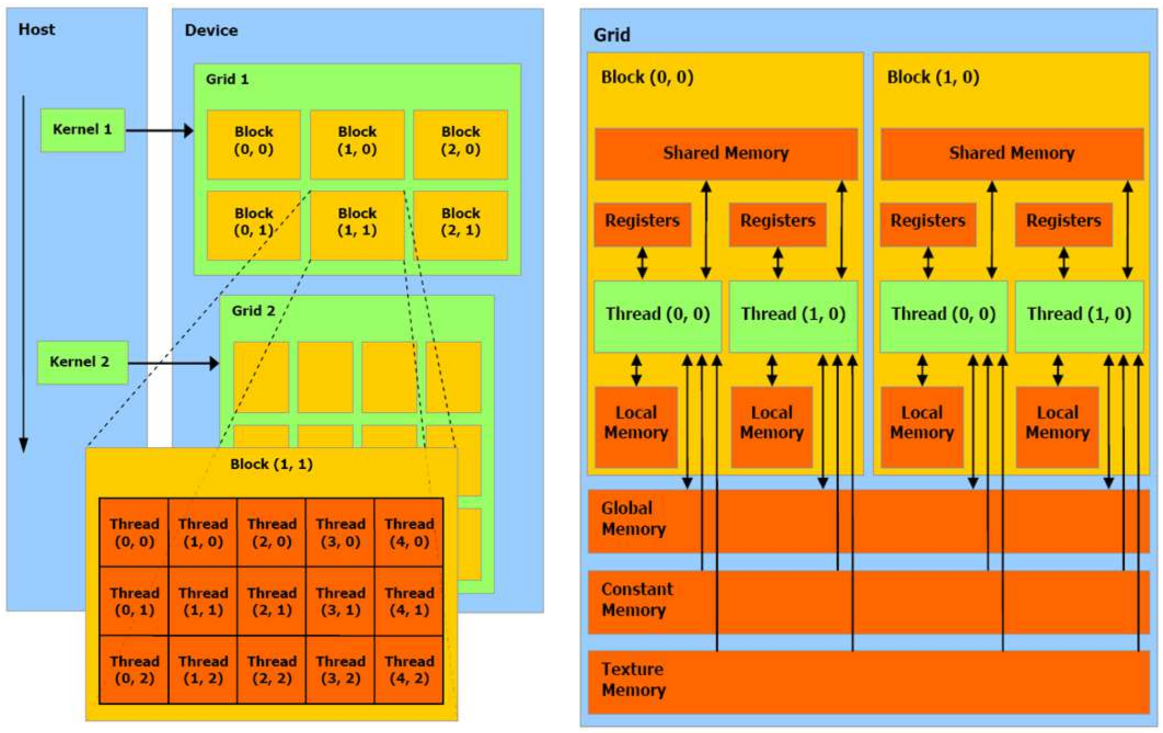


Figure 4.1: Architecture of CUDA's threads and memories hierarchies. *Left side.* Threads organization: a kernel is invoked from the CPU (the host) and is executed in multiple threads on the GPU (the device). Threads are organized in three-dimensional structures named blocks which are, in turn, organized in three-dimensional grids. The programmer must decide the dimensions of blocks and grids before the kernel launch. *Right side.* Memory hierarchy: threads can access data from multiple kind of memories, all with different scopes and characteristics: registers and local memories are private for each thread; shared memory let threads belonging to the same block communicate, and has low access latency; all threads can access the global memory, which suffers high latencies, but it is cached since the introduction of the Fermi architecture; texture and constant memory can be read from any thread and are equipped with a cache as well. Figures are taken from Nvidia's CUDA programming guide [244].

GPUs are equipped with different types of memory. As described in Figure 4.1, right side, the GPU memory hierarchy consists in the *global memory* (accessible from

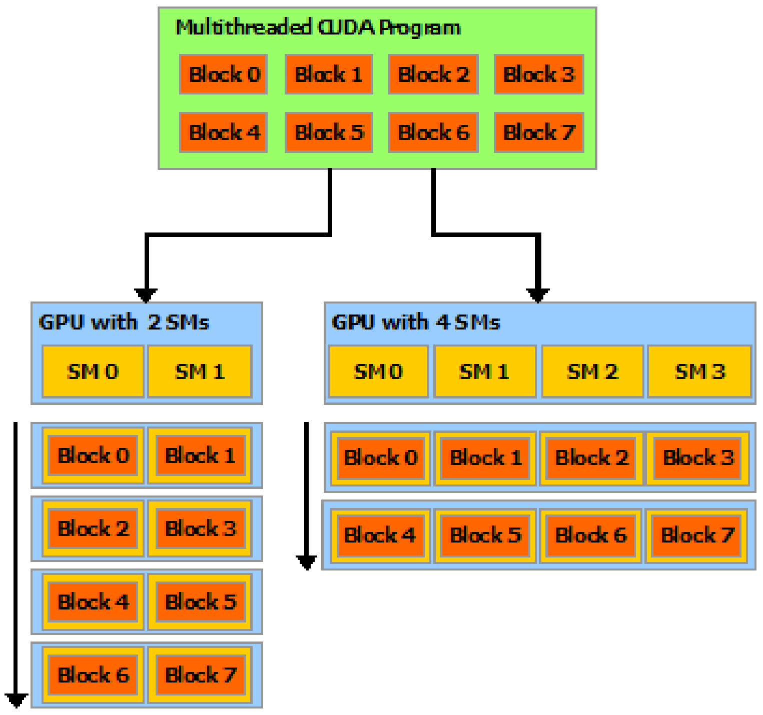


Figure 4.2: CUDA automatic scalability: blocks are automatically scheduled on the available streaming multiprocessors (SM), in a transparent way for the programmer. Figure taken from [244].

all threads), the *shared memory* (accessible from threads of the same block), the *local memory* (registers and arrays, accessible from owner thread), and the *constant memory* (cached and not modifiable). The best performances in the execution of CUDA code can be achieved by exploiting the shared memory as much as possible. Unfortunately, the shared memory is a very limited resource (i.e., 49152 bytes for each multi-processor, since the introduction of the Fermi architecture) that introduces restrictions on the blocks' size. On the contrary, the global memory is very large (thousands of MBs) but suffers of high latencies. In order to mitigate this drawback, the global memory has been equipped with a L2 cache (see Figure 4.3), starting from the Fermi architecture. Moreover, with this architecture the programmer can balance 64 KB of fast on-chip memory between the shared memory and L1 cache, specifying two different configurations: 48 KB for the shared memory and 16 KB for L1 cache, or 16 KB for the shared memory and 48 for L1 cache. In addition, using the Kepler architecture, a third and perfectly balanced configuration can be specified by assigning the same amount of memory (32 KB) both to the shared memory and L1 cache.

Since blocks are asynchronously scheduled on different SMXs, there exist no communication nor synchronization primitives between blocks. Intra-block communication and synchronization are performed by means of the shared memory and the family of `__synchthreads()` kernel functions, respectively. A special version of this function, namely `__synchthreads_count(int p)`, evaluates simultaneously the predicate `p` for all threads of a block and returns, to all threads in that block, the number of threads for which the predicate `p` is different from zero.

An implicit synchronization of blocks can be achieved by launching two consecutive kernels, which are queued for execution (see Figure 4.1, left side). Block-level concurrency can be performed by means of CUDA *streams*. Since they go beyond the scope of the present thesis they will not be discussed any further.

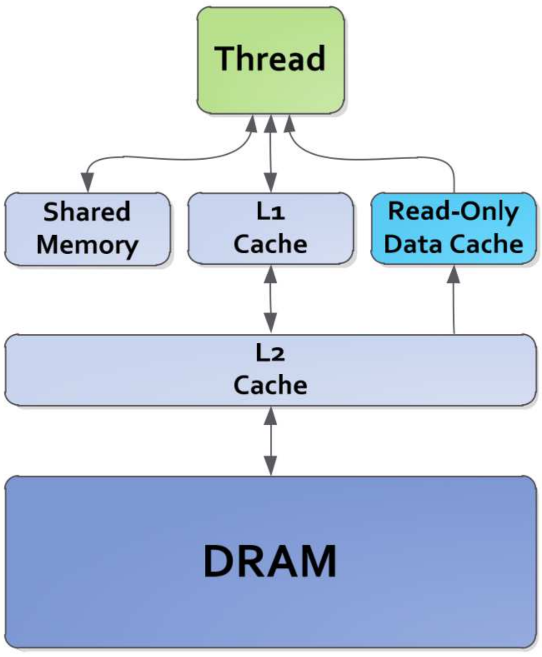


Figure 4.3: Schematic description of the memory hierarchies in Fermi and Kepler architectures. GPUs based on these architectures are equipped with a two level data cache and a read-only data cache. The shared memory and the L1 cache share the same on-chip 64 KB memory banks; the amount of memory can be reconfigured by the user, according to the specific needs of the application. Figure taken from Nvidia's Kepler GK110 whitepaper [245].

4.2 Random numbers generation in CUDA

Stochastic simulation and optimization algorithms largely rely on random numbers generators (RNGs). Nvidia's software development kit [244] contains several libraries and utilities that help developers in the process of creating software for this architecture. CURAND is a RNG library which allows the GPU-based generation of random deviates that can be used both by the host (via memory copy) or directly by the device.

The latest version of the CURAND library, at the time of writing, is available in the CUDA toolkit v6.0 [246] and gives access to many different RNGs that can produce both pseudo- and quasi-random sequences. Since quasi-random sequences are not exploited in any of the tools described in this thesis, they will not be discussed.

The pseudo-random sequences implemented in CURAND are the following: XORWOW [200], MRG32K3A [177], PHILOX4-32 [196], and two versions of Mersenne Twister (MT) [204]. Among these RNGs, MT yields the longest pseudo-random sequences thanks to its 2^{11213} period; PHILOX4-32 generates sequences with a period equal to 2^{128}; XORWOW and MRG32K3A generate sequences of pseudo-random numbers with a period around 2^{160} and 2^{191}, respectively. MT was not used for the implementation of the methodologies of this thesis because it has three drawbacks: (*i*) at most 256 threads per block can operate simultaneously [242], (*ii*) the memory footprint is larger than the other generators [225], (*iii*) it is much slower than the other algorithms.

XORWOW is faster than MRG32K3A, but it is known to present statistical flaws [133] and it is rejected by 3 of the 106 tests of the BigCrush statistical test suite [176]. PHILOX4-32 has the smallest period, even though it allows to produce up to four 32-bit random numbers at once. Since MRG32K3A represents the best trade-off between performances, period length and does not have the aforementioned drawbacks of MT, it has been exploited in all implementations of thesis.

CURAND exploits these RNGs to generate random deviates with uniform, standard normal and log-normal distributions; since the introduction of the CUDA toolkit v5.0, CURAND libraries also offer the possibility to generate the Poisson-distributed random deviates, which are required by the tau-leaping algorithm described in Sections 2.3.2 and 5.3.

4.3 Computational Biology and general-purpose GPU computing

The *in silico* analysis of biological systems can be hard because, in general, it is extremely computationally expensive. For instance, in the case of the biochemical systems described in Chapter 2, the running time of simulations can be prohibitive, even in the case of small size systems. This does not represent a problem in itself, but can become a relevant issue because some analysis methodology (e.g., parameter sweep analysis [51], parameter identifiability [273], parameter estimation [216], reverse engineering [167], sensitivity analysis [293]) requires a large number of simulations to determine the result.

Since simulations are mutually independent, they can be straightforwardly accelerated by means of dedicated parallel architectures [20] (e.g., GRID computing, clusters of computers) which come with a cost in energy, money and maintenance. On the contrary, GPUs are cheap, power-efficient and widespread in common desktop computers, so that they are becoming attractive alternatives for the implementation of tools for systems biology [71], bioinformatics [313, 365], computational biology [123, 300, 323] and chemistry (e.g., molecular dynamics [124] and quantum chemistry [111]).

Focusing on the biochemical simulation, for instance, Ackermann *et al.* developed a tool for the automatic conversion of SBML[1] files into CUDA executables [3], able to simulate multiple instances of a system by means of ODE by means of ODEs numerical integrators (Section 2.3.1). Li and Petzold implemented the SSA algorithm (Section 2.3.2) on GPUs, showing a relevant speedup with respect to the sequential execution [182]. In this context, this thesis will provide two examples of GPU-powered deterministic [230] and stochastic simulators [235], described in detail in Chapter 5.

The parallelization of multiple simulations (*coarse-grain acceleration*) is not the only way to leverage the power of GPUs. For instance, Komarov *et al.* proposed a *fine-grain* implementation of the tau-leaping algorithm (Section 2.3.2) [165], in which each thread is assigned to a specific reaction. Unfortunately, coarse grain implementations generally suffer from communication and synchronization between all threads, which are necessary to calculate some intermediate values (e.g., the putative time step τ in stochastic simulations).

GPUs were also exploited for spatial simulation (Section 2.2.6), for instance by extending existing agent-based simulators to support GPUs [79, 279, 280], or by distributing calculations of subvolumes-based methods [250] or lattice-based methods [281].

[1]The Systems Biology Markup Language is a standard XML-based exchange format for biochemical models [141].

Part II

Novel Work

Chapter 5

GPU-accelerated biochemical simulation

Many tasks that are typical of Systems and Synthetic Biology require a massive number of simulations of the system under investigation. In particular, the evolutionary methodologies for parameter estimation, reverse engineering and automatic design described in this thesis (Chapters 6, 7 and 8, respectively) require a large number of simulations to assess the fitness values of candidate solutions (see Chapter 3 for further details). Simulations have a huge impact on the execution time of these methodologies but, since they are mutually independent, they can be straightforwardly accelerated by means of a parallel architecture, for instance the GPGPU computing described and motivated in Chapter 4. Thus, the first step of this thesis concerned the development of efficient GPU-powered simulators.

In this chapter, three different *coarse-grain* GPU-accelerated simulators of RBMs are described: a deterministic simulator exploiting the law of mass-action (cupSODA, Section 5.1); a deterministic simulator, extending cupSODA and tailored for the simulation of a model of the blood coagulation cascade (coagSODA, Section 5.2); a stochastic simulator based on tau-leaping (cuTauLeaping, Section 5.3). Besides their use for fitness evaluations, the main goal of these simulators is the analysis of biochemical models, to predict the way the system behaves in normal or perturbed condition. As a matter of fact, the intensive exploration of high-dimensional parameter spaces, corresponding to different system conditions, allows to devise the different emergent behaviors that the system can present [9, 59, 355].

The main contents of this chapter are published in journal papers [51, 230, 235] and conference proceedings [22, 233].

5.1 Deterministic biochemical simulation: cupSODA

cupSODA is a GPU-powered simulator for biological systems that allows to efficiently execute a large number of parallel deterministic simulations at a considerable reduced computational cost with respect to CPUs. In particular, cupSODA exploits the massive parallelism of CUDA architecture to execute independent deterministic simulations in each thread.

In the case of deterministic models, a system of ODEs is used to describe how the concentration of each chemical species varies in time, as described in Section 2.3.1; according to the law of mass action [226], these equations can be derived from a given mechanistic RBM of the biological system under investigation (see Section 2.2.3).

cupSODA was conceived as a "black-box" simulator for models based on MAK, whose peculiarity is that it automatically derives the system of ODEs — and perform their numerical integration using the LSODA algorithm [257] — simply starting from a set of biochemical reactions which describe the molecular interactions between all the species occurring in the system. This way, any user — having or not either GPU programming skills or mathematical modeling expertise — is able to run parallel deterministic simulations at reduced computational costs. Indeed, when a large number of mutually independent simulations has to be performed, LSODA turns out to be very time consuming if the simulations are run in a sequential manner on the CPU. In what follows it is shown how a GPU implementation of LSODA turns out to be extremely advantageous.

5.1.1 GPU implementation of cupSODA

cupSODA relies on a C version of the LSODA ODE solver [257] (see Section 2.3.1), ported and adapted to the CUDA architecture in order to be run on the GPU. cupSODA was designed to be a cross-platform implementation, so that it can be run on Microsoft Windows, Linux, and Apple OSX based operating systems. Nevertheless, to exploit CUDA's massive parallelism for the execution of different and independent simulations in each thread, a Nvidia GPU is required; therefore, cupSODA itself requires a Nvidia video card to be executed.

In [257], LSODA was designed to solve differential systems in the canonical form, whereby the user is supposed to specify the system of ODEs, that must be implemented as a custom C function that is then passed to the algorithm. In order to speed up the computation when dealing with stiff systems, the Jacobian matrix must be implemented as a custom C function as well.

cupSODA, on the contrary, was designed to the purpose of being a *black-box* simulator, that can be easily used without any programming skills. In particular, cupSODA consists in a tool that automatically converts the RBM of a biological system — which can be indifferently given according to a deterministic or a stochastic formulation[1] — into the corresponding system of ODEs, in conformity with the MAK [358], and then it encodes the ODEs and the corresponding Jacobian matrix as arrays.

These two arrays are loaded into the GPU, then parsed and interpreted as custom functions on the device side. In order to encode each term of each equation into a linear data structure, without any loss of information, both arrays contain the following data:

- a signed multiplicative factor of the term;
- the index of the kinetic constant associated to the term;
- the number of the chemical species involved in the term and their corresponding indices (see an example in Figure 5.1).

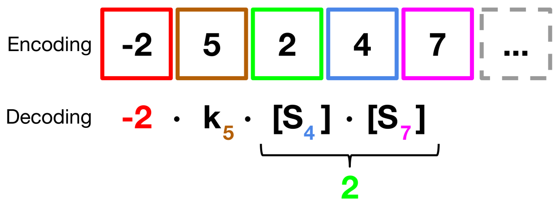


Figure 5.1: Example of the encoding methodology in cupSODA, showing the internal representation of a single term of an ODE, used for the parsing device-side. From the left to the right, the values of the array have the following semantics: the multiplicative factor of the term (red); the index of the kinetic constant associated to the term (brown); the number of species involved in this term (green) and the indices of these species (blue and violet). All the terms of all the ODEs are then linked together in a single one-dimensional array.

The terms of all ODEs are linked together in these arrays. To efficiently parse the arrays inside the GPU, cupSODA uses two additional arrays storing the offsets of each equation (i.e., the indices of each term of each equation); the parsing algorithm consists in the function given in Figure 5.2. A similar function is defined to parse the Jacobian matrix.

[1]cupSODA automatically executes the conversion from the stochastic to the deterministic formulation of both reaction constants and initial molecular amounts, given that the volume of the modeled biological system is specified.

```
function decode_ODEs ( encoded_ODE, ODE_offset ):
  dX = [0, ..., 0]
  position = 0
  for i = 0 to ODE_offset.length do:
    ode_value = 0
    while( position<ODE_offset[i] ) do:
      term = encoded_ODE[position]
      term *= k[encoded_ODE[position+1]]
      species = encoded_ODE[position+2]
      for s = 0 to species do:
        term *= X[encoded_ODE[position+3+s]]
      end for
      position += (3+species)
      ode_value += term
    end while
    dX[i] = ode_value
  end for
  return dX
end function
```

Figure 5.2: Pseudocode of the parsing algorithm, exploited device-side by cupSODA for the decoding of ODEs. A similar code is used for the decoding of the Jacobian matrix.

cupSODA is able to launch multiple threads which run independent parallel simulations of the same model, with each thread exploiting its own parameterization and initial conditions. So doing, large numbers of simulations in perturbed conditions can be executed with reduced running times. To this aim, parameters and initial conditions are contained in coalesced arrays, a strategy that allows a faster fetching of data from the global memory.

Besides the global memory (accessible from all threads), cupSODA also exploits the shared memory (accessible from threads belonging to the same block), the local memory (registers and arrays, accessible from owner thread), and the constant memory (cached and not modifiable). In particular, being the data transfer between host and device very time consuming, all temporary results are allocated on the memories of the GPU, and they are read back as soon as the simulations are over. To obtain a further reduction of the memory latencies, the current state and time of each simulation are stored into the shared memory, while all the constants values (e.g., number of reactions and chemical species in the model, length of ODEs and Jacobian arrays) and LSODA settings are stored into the constant memory.

Since the amount of shared memory is limited on each SMX — and poses a limitation on the blocks' size — cupSODA automatically calculates the number of threads per block and blocks per grid. Double precision floating point values are exploited to allocate the states of the system and the current time, so that the consumption of shared memory of each thread is $SH = 8 \times (N+1)$ bytes of shared memory during the ODEs integration, where N is the number of chemical species in the system. It is possible to determine the threads-per-block value as $T_{pb} = \left\lfloor \frac{MAX_{shared}}{SH} \right\rfloor$, where MAX_{shared} is the shared memory available on the GPU, so that the number of resulting blocks is $B_{pg} = \left\lfloor \frac{U}{T_{pb}} \right\rfloor$, where U is the total number of threads requested by the user.

LSODA requires additional parameters for its functioning, the most relevant being the absolute and relative error tolerance values and the maximum number of internal steps for each integration interval. AET and RET values can be either scalar values or specific vectors for each ODE: cupSODA accepts all combinations. Moreover, these values can be specified for each individual thread, allowing the user to simulate the same system with different tolerances and compare their outcomes.

cupSODA can also be easily employed to compare the outcome of simulations with any available experimental data (e.g., the concentration of some chemical species measured at sampling instants $t_0, \ldots, t_\iota$), for instance to validate the model under investigation or to perform a parameter estimation. To this aim, cupSODA invokes the LSODA kernel $\iota - 1$ times: each time the kernel is run over a (simulated) time interval of length $\Delta = t_i - t_{i-1}$, $i = 1, \ldots, \iota$, and the concentration values of the output species are stored at the end of each Δ. Once the concentrations are stored, cupSODA provides a set of metrics (e.g., root mean square, relative distance) that the user can exploit to evaluate the distance between each simulation outcome and the target dynamics.

5.1.2 Results

To show cupSODA's effectiveness, its performances were compared against the COmplex PAthways SImulator (COPASI [140]), which is considered here as the reference sequential simulator. To show the suitability of cupSODA, different batches of independent deterministic simulations were performed, using three biological models characterized by an increasing complexity, described in Appendix A:

- the Michaelis-Menten (MM) enzymatic kinetics [226];
- a simple model of gene expression in prokaryotic organisms (PGN) [349];
- the signaling pathway Ras/cAMP/PKA in the yeast *S. cerevisiae* [27].

During each test, the dynamics of all chemical species were stored, each dynamic consisting of 100 time instants — uniformly sampled over the whole simulation time — keeping track of the overall running time.

The GPU used for the tests is a Nvidia GeForce GTX 590, a video card with Fermi architecture equipped with 2×16 streaming multiprocessors for a total of 1024 cores. The performances of this GPU were compared with a quad-core CPU Intel Core i7-2600 with a clock rate of 3.4 GHz. A direct comparison of GPUs and CPUs is a hard task, because of their architectural differences (e.g., the multiple cache levels of CPUs). Moreover, nominal peak capabilities of GPUs (2.48 TFlops in single precision, in the case of GTX 590) can be achieved only by implementing kernels completely adhering to the underlying SIMD architecture (e.g., by avoiding any conditional branch in the code). In addition, the occupancy of GPUs is affected by the usage of registers and shared memory, which are both limited resources on each streaming multiprocessor. cupSODA suffers from a high register pressure, which poses a limitation on the number of simultaneous blocks that can be executed by each multi-core processor: currently, cupSODA's implementation is limited to 2 blocks on Fermi GPUs and 4 on Kepler GPUs. Hence, the theoretical computational power peak is hard to be reached; nonetheless, the performances of these two hardware devices are compared here, since they are well representative of hardware components typically found in personal computers. All tests were run on a system running the O.S. Microsoft Windows 7, CUDA version 5.0, COPASI 4.8 (build 35).

Figure 5.3 shows a comparison of the overall running times of COPASI and cupSODA, obtained by performing an increasing number of simulations with LSODA for the three test models. The results show that cupSODA largely outperforms the LSODA algorithm implemented in COPASI and, in the case of 10^5 simulations (i.e., parallel threads) for the MM model (Figure 5.3, top), the computational cost on the GPU is nearly two orders of magnitude smaller than the CPU, namely, 3.358 sec *vs.* 289.335 sec, which corresponds to a 86× speedup. In the case of the PGN model (Figure 5.3, middle), the execution of 10^4 simulations takes 43.821 sec on the CPU, while it takes just 0.391 sec on the GPU, resulting in a 112× speedup. Vice versa, a small number of simulations does not yield better performances, since the running time for 10 simulations of this model is similar on the two architectures: 0.047 sec (CPU) *vs.* 0.046 sec (GPU). The execution of 10^5 simulations of the Ras/cAMP/PKA model (Figure 5.3, bottom) takes 6133.3 sec on the CPU and just 268.58 sec on the GPU, resulting in a 23× speedup. This result clearly states that cupSODA is convenient

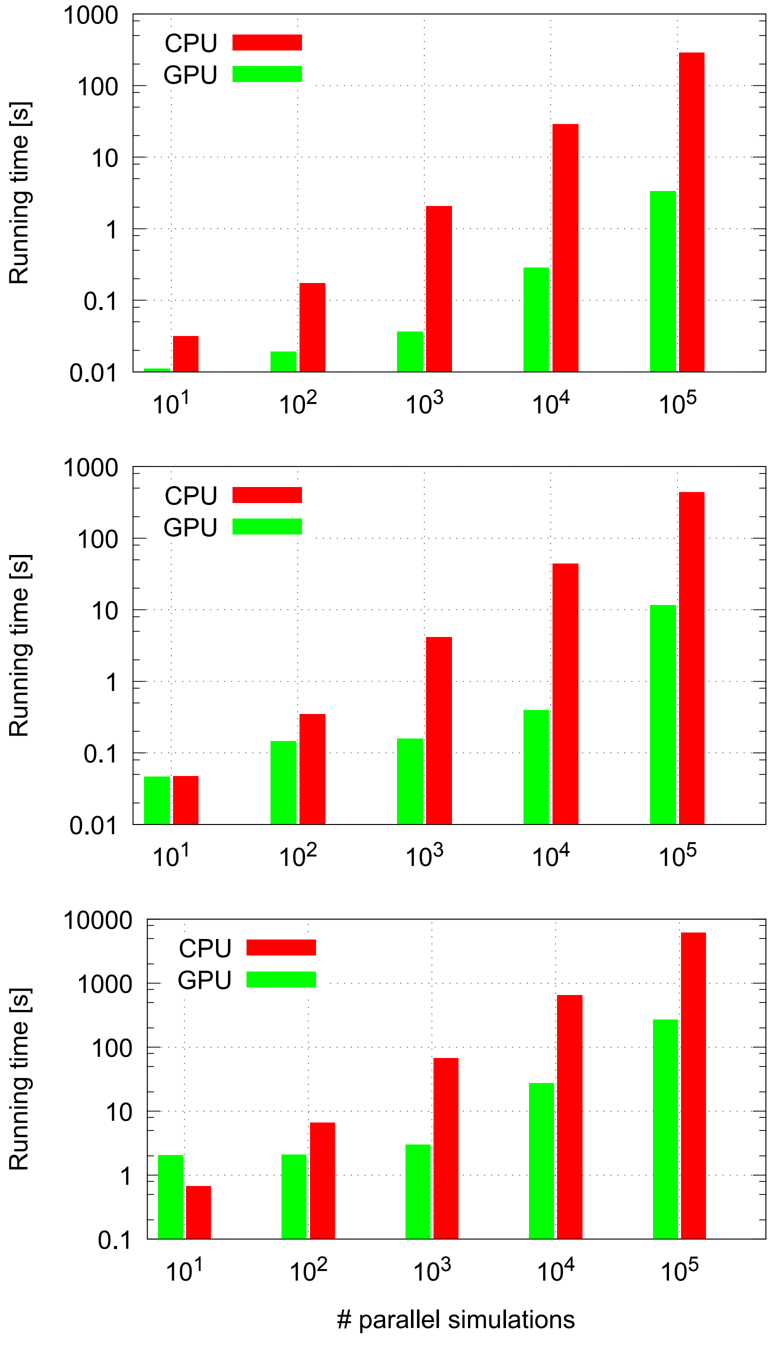


Figure 5.3: Comparison between the computational time of cupSODA (green histograms) and COPASI (red histograms) for simulating the MM model (top), the PGN model (middle), and the Ras/cAMP/PKA model (bottom). The y-axis are in logarithmic scale; RET= 10^{-10}, AET= 10^{-10} for MM and PGN models, RET= 10^{-10}, AET= 10^{-14} for the Ras/cAMP/PKA model. For all tests, the maximum number of internal steps allowed during each call of LSODA was set to 10000.

when more than 10 simulations are to be executed, since the computational cost for a small number of simulations can be lower on the CPU.

The impact of the use of different memories on cupSODA was also investigated, by executing 10^5 simulations of the Ras/cAMP/PKA model exploiting either the shared memory or the global memory to store the current state and time of the simulations. The running times were, respectively, 27.501 sec and 54.93 sec: cupSODA is twice as fast as the naïve porting of LSODA when the low latency memories are intensively exploited; in addition, it is worth noting that these running times are always lower than the CPU (651.741 sec).

5.1.3 Discussion

cupSODA was not the first attempt in porting LSODA to the CUDA architecture. A previous GPU implementation of LSODA algorithm was proposed in the cuda-sim library [370], a Python package providing GPU-accelerated biochemical simulations. Nevertheless, the aim and design of cuda-sim are very different from cupSODA, as the latter allows to perform a massive number of simulations without writing any source code. cuda-sim also relies on a *just-in-time* technique, whereby the code for LSODA that will be executed on the GPU is automatically created and compiled at run-time: this is indeed a flexible and elegant solution, but adds a relatively long compilation time and requires the availability of a CUDA compiler and the CUDA toolkit on the running machine.

During the development of cupSODA it was chosen the encoding of ODEs and the Jacobian matrix into linear arrays which are processed device-side, without the need for any intermediate recourse to the CUDA driver API or any meta-programming techniques. Hence, with cupSODA it is possible to immediately simulate any biological system modeled according the MAK, a characteristics that is particularly appealing when the model itself is repeatedly modified, for instance when it undergoes a reverse engineering process [59, 167, 234], for example when running cuRE (Chapter 7): in such a case, cupSODA only needs to update the GPU representation of the ODEs and the Jacobian matrix, in order to be ready to run a massive number of simulations of the new model.

cupSODA becomes more convenient than the CPU counterpart when a relevant number of parallel simulations has to be run, with a break-even that depends on the complexity of the system under investigation. Indeed, when performing demanding computational analysis (e.g., parameter sweep, parameter estimation or sensitivity analysis) the outstanding advantage of novel softwares as cupSODA clearly comes to light. Moreover, cupSODA's fitness evaluation functions allow a direct integration into the PE, RE and ED methodologies described in the following chapters.

5.2 Deterministic simulation of the Blood Coagulation Cascade: coagSODA

cupSODA relies on models based on MAK, which is the most general framework for the description of biochemical kinetics. This choice has sound chemical-physical justification, but sometimes alternative kinetic functions are considered to describe specific biochemical phenomena. For instance, Hill kinetics is largely exploited to model the cooperative binding of ligands to macromolecules, which is enhanced by the presence of other ligands on the molecule [226], as in the case of oxygen's binding to haemoglobin [132].

In this section it is presented coagSODA, a simulator based on cupSODA specifically designed for the analysis of a model of the Blood Coagulation Cascade (BCC), which integrates MAK and Hill kinetics. As it is shown hereby, coagSODA allows to efficiently execute a large number of parallel deterministic simulations at a considerable reduced computational cost with respect to CPUs [51].

5.2.1 The BCC model

Blood is an essential component in human life, whose primary functions are to feed cells by delivering a multitude of nutrients, including oxygen, that is necessary for cellular respiration, and to carry away the cellular wastes, such as carbon dioxide. Specialized cells and fluids in blood perform many physiological functions, and can be isolated and analyzed through specific laboratory tests, giving the opportunity to settle a person's health condition. Thanks to its key role in making diagnosis of numerous diseases, blood is the subject of an intense scientific research [283]. All blood components are kept within appropriate concentration ranges by means of fine-tuned regulatory mechanisms, ruled by several feedback controls; the constancy of blood composition is maintained thanks to the circulation through an intricate network of vessels. In particular, humans have evolved a complex hemostatic system that, under physiologic conditions, is able to maintain blood in a fluid state; however, in response to any vascular injury, this system can rapidly react and seal the defects in the vessels wall in order to stop the blood leakage [62]. Indeed, the circulatory system is self-sealing, otherwise a continuous blood flow from even the smallest wound would become a threaten for the individual's life.

These regulation processes are governed by means of the BCC, a complex network of cellular reactions which, under physiological conditions *in vivo*, are inhibited by the presence of intact endothelium [213]. In order to allow blood coagulation, in humans there exist 13 blood clotting proteins, called *coagulation factors*, which are usually designated by Roman numerals I through XIII. Following a vascular injury, platelets become active and the tissue factor (TF, or factor III) is exposed in the subendothelial tissue, starting the BCC. The ultimate goal of BCC is to convert prothrombin (factor II) into thrombin (factor IIa - that is, the activated factor II), the enzyme that catalyzes the formation of a clot. Traditionally, the BCC is divided into the *extrinsic* and *intrinsic pathways*, both of which lead to the activation of factor X [343]. The last part of the cascade, downstream of this factor, is called the *common pathway* and leads to the formation of fibrin monomers, whose polymers finally constitute the backbone of the clot.

Excluding thrombin, all the enzymes involved in blood clotting are characterized by a low activity, which increases upon binding to a specific protein cofactor (e.g., factors V and VIII) or to appropriate phospholipid surfaces (e.g., the plasma membranes of active platelets) [343]. In BCC pathways, there exist also inhibitory factors, which limit the activity of the various active proteases and, therefore, allow to regulate the whole reactions cascade. When the hemostatic system is unregulated, thrombosis (i.e., the formation of a blood clot obstructing the blood flow in vessels) may occur due to an impairment in the inhibitory pathway, or because the functioning of the natural anticoagulant processes is overwhelmed by the strength of the hemostatic stimulus [62].

In order to investigate the individuals' variations in blood coagulation components, and the corresponding response to perturbed conditions, it is necessary to define a mathematical model of the BCC. The model presented in [56], built upon a previous model [136], describes the intrinsic, extrinsic and common pathways and, more importantly, it accounts for the platelet activation, as well as the presence of several inhibitors (e.g., the tissue factor pathway inhibitor (TFPI), antithrombin III and C1-inhibitor). This approach allows, for instance, to study the alterations (prolongation or reduction) of the time necessary to form the clot (i.e., the *clotting time*).

The BCC model used as a basis for coagSODA is a slightly reduced version of the deterministic model given in [56], where a small set of reactions downstream of the clot formation were excluded since they do not have any effect on the clotting time. A graphical sketch of the BCC model, which overall consists in 96 reactions among

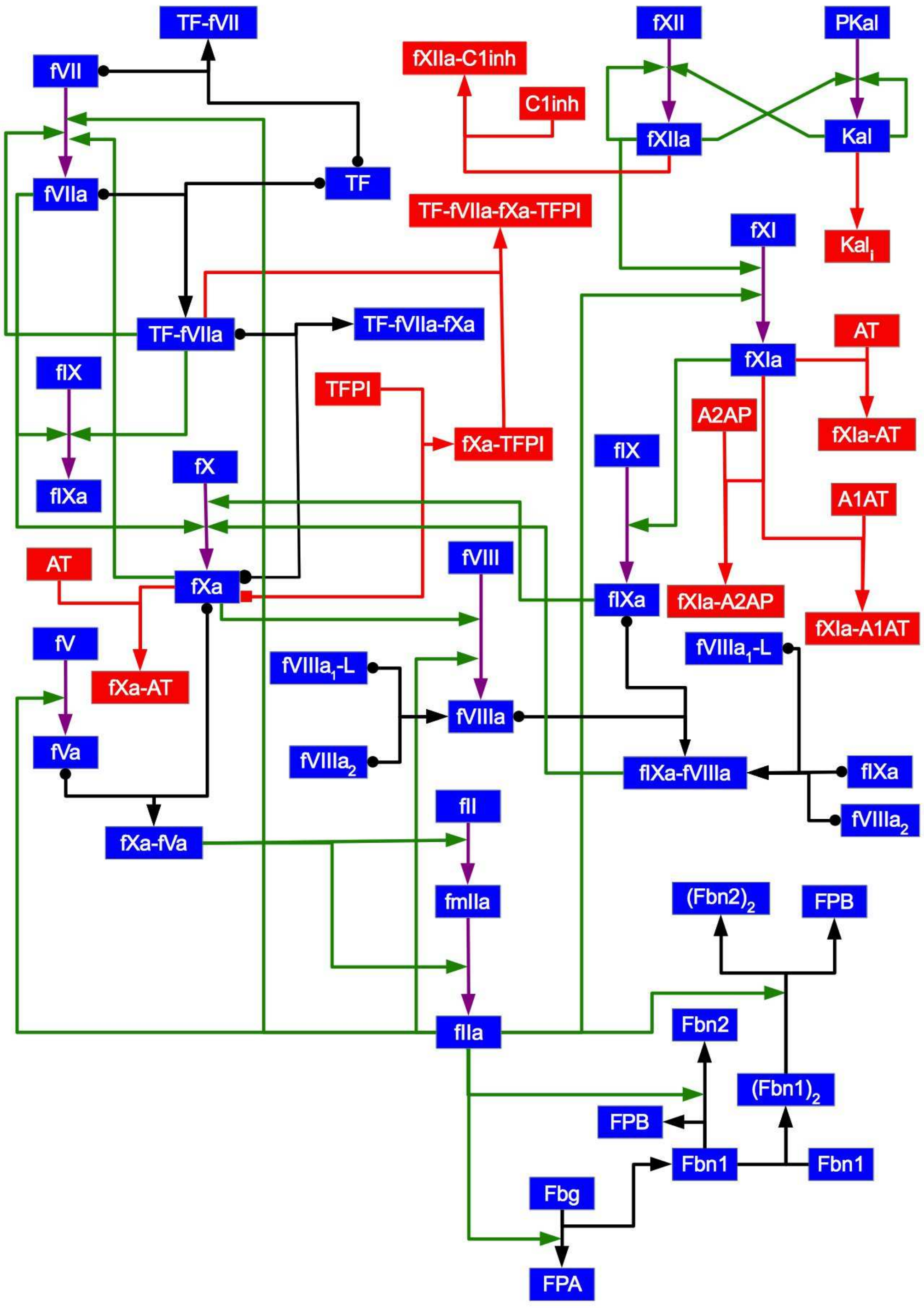


Figure 5.4: Graphical representation of the BCC model considered in this work. *Legenda.* Blue box: coagulation factor; red box: inhibitor and related complexes. Black arrow: complex formation; green arrow: catalytical activation; violet arrow: activation; red arrow: inhibition. The reaction is reversible if the arrow tail consists in a dot.

71 molecular species, is given in Figure 5.4. The RBM and the initial state of this system are reported in Appendix A.4. The system of ODEs, needed to carry out the simulations and the parameter sweep analysis (PSA) presented in what follows, is based on MAK, except for 14 reactions that are influenced by a specifically defined Hill function. The rationale behind the use of Hill functions in the BCC model relies on the necessity of modeling the physiological levels of thrombin concentration as a function of platelet activation, as thoroughly described in [56]. Namely, the platelet activity is mimicked by modulating the rate of the dissociation of the complexes formed on a platelet's surface by means of a specific factor ε, which is calculated with a special equation during the integration steps (see Section 5.2.2).

coagSODA circumvents the need of manually defining the system of ODEs that describe the BCC network. More precisely, coagSODA is able to automatically derive the system of (MAK and Hill function-based) ODEs — and then to perform their numerical integration by means of the LSODA algorithm (Section 2.3.1) — starting from the given set of 96 biochemical reactions, which fully describe the molecular interactions between all the species involved in the BCC.

5.2.2 GPU implementation of coagSODA

coagSODA realizes the run-time calculation of the ε value required to correctly simulate the activity of platelets; the variable ε depends on a Hill function that was fit against experimental data, which quantify the platelet activation status (see details in [56]). The value of ε influences the formation of some complexes within the BCC, occurring on the platelets' surface; namely, ε intervenes on their dissociation constants by modifying the activity of reactions of the form

$$A + B \underset{k_2}{\overset{k_1}{\rightleftharpoons}} AB,$$

so that the corresponding standard ODEs are changed to yield new equations of the form:

$$\frac{\mathrm{d}[AB]}{\mathrm{d}t} = k_1 \cdot [A] \cdot [B] - \frac{k_2 \cdot [A] \cdot [B]}{\varepsilon}. \tag{5.1}$$

The set of 14 reactions of the BCC model influenced by ε is given in given in Appendix A.4.

The value of ε in Equation 5.1 depends on the following Hill function $\mathfrak{f}$, which quantifies the state of platelets activation according to the thrombin concentration (denoted as $[IIa]$), that is, the factor catalyzing the formation of the fibrin clot:

$$\mathfrak{f}([IIa^*(t)]) = \frac{[IIa^*(t)]^{1.6123}}{[IIa^*(t)]^{1.6123} + (2.4279 \cdot 10^{-9})^{1.6123}},$$

where $[IIa^*(t)] = \max_{t' \in [0,t]}\{[IIa(t')]\}$, for a chosen time interval $[0, t]$ of simulation.

The value $[IIa^*(t)]$ represents the maximum transient thrombin concentration, it is needed to simulate the fact that in physiological conditions the thrombin concentration starts decreasing after monotonically rising to a peak (Figure 5.5). So doing, $[IIa^*(t)]$ never decreases once it reaches its maximum magnitude: $[IIa^*(t)]$ is equivalent to $[IIa(t)]$ until the concentration of factor IIa reaches the peak, while thereafter it remains constant at that value, which is the maximum in the considered time interval $[0, t]$. Function $\mathfrak{f}$ allows to simulate the physiological condition whereby platelets remain active also when the thrombin concentration decreases.

For a given concentration of factor IIa, the maximum platelet activation state ε_{max} is defined as:

$$\varepsilon_{max} = \varepsilon_{max_0} + (1 - \varepsilon_{max_0}) \cdot \mathfrak{f}([IIa^*(t)]), \tag{5.2}$$

where ε_{max_0} defines the basal activation state of the platelet at simulation time $t = 0$. The value ε_{max_0} is initially set to 0.01, assuming a basal 1% binding strength of coagulation factors to the resting platelet surface. When the full activation of platelets is reached, ε_{max} is equal to 1 and the complex dissociation constants are minimized [56].

coagSODA calculates at each time instant the platelet activation state ε by solving the following differential equation:

$$\frac{\mathrm{d}\varepsilon}{\mathrm{d}t} = k(\varepsilon_{max} - \varepsilon),$$

where the constant k is inversely proportional to the time scale of platelets activation and is set to 0.005. This is consistent with the fact that platelets do not instantly achieve their maximum attainable activation state (ε_{max}), but they reach it on a physiologically relevant timescale.

In dealing with the 14 reactions of the BCC model that are influenced by the Hill function, the value of $[IIa^*(t)]$ must be stored in the GPU because, during each integration step, coagSODA recalculates Equation 5.2, which exploits the value $[IIa^*(t)]$ to determine a new ε value. Since the CUDA architecture does not offer static

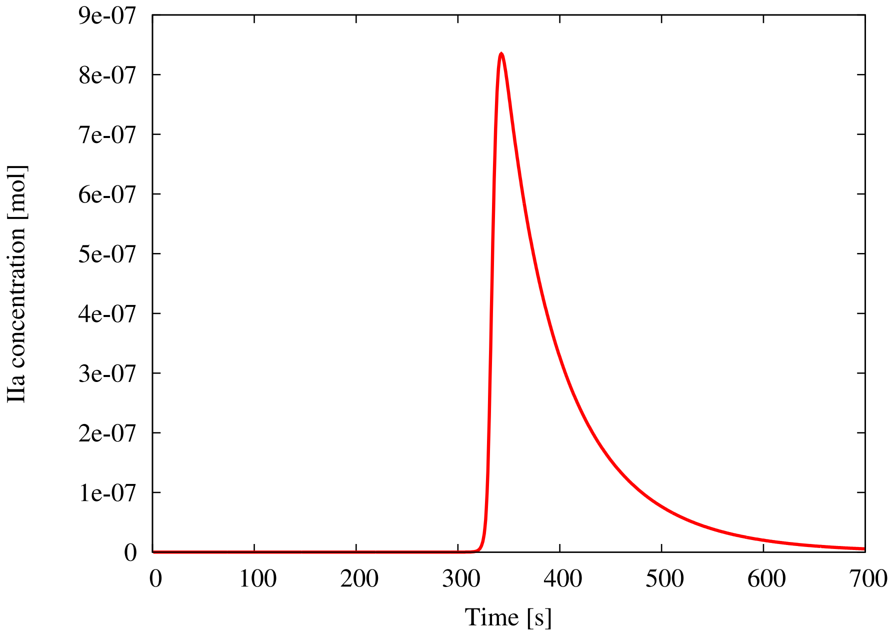


Figure 5.5: Dynamics of thrombin in physiological condition.

variables[2], the information for each thread has to be memorized in the global memory. The accesses to the global memory and the computational costs due to the additional calculations slow down the integration process, with respect to the integration of ODEs performed according to a strictly MAK (as in the case of cupSODA). Nevertheless, in Section 5.2.3 it is shown how this parallel implementation largely outperforms a sequential counterpart.

5.2.3 Results

In this section the results of the PSA carried out on the BCC model are discussed. This analysis is useful to investigate either the prolongation or the reduction of the clotting time in response to perturbed values of some reaction constants and of the initial concentration of some molecular species, chosen according to their meaning within the whole pathway. PSA was performed by generating a set of different initial conditions — corresponding to different parameterizations of the model — and then automatically executing the deterministic simulations with coagSODA. The use of GPU technology is fundamental in this type of analysis, especially for large biological systems as in the case of BCC, because it drastically reduces the computational time.

[2]In traditional CPU-bound C programming, static variables have a lifetime that extends across the entire run of the program, but they have a scope that is limited to the function they were declared in.

The sweep analysis for single parameters (PSA-1D) was performed considering a logarithmic sampling of numerical values of each parameter under investigation (reaction constant or initial molecular concentration) within a specified range with respect to its physiological reference value. The sweep analysis over pairs of parameters (PSA-2D) was performed by simultaneously varying the values of two parameters within a specified range, considering a logarithmic sampling on the resulting lattice. The logarithmic sampling allows to uniformly span different orders of magnitude of the parameters value using a reduced and fine-grained set of samples, therefore efficiently analyzing the response of the system in a broad range of conditions.

To determine the response of the BCC to perturbed conditions, we chose the *clotting time* (CT) as output of the PSA. The CT is defined as the time necessary to convert the 70% of the fibrinogen into fibrin (see Figure 5.6), conventionally assumed to correspond to the time required to form the clot [87], and it is generally used in laboratory tests for monitoring the therapy with anticoagulant drugs. According to the model defined in [56], the reference value of CT is around 300 sec in physiological conditions. The response of the BCC was investigated by PSA by evaluating the CT in various conditions, corresponding to different values of the reaction constants, varied over six orders of magnitude with respect to their physiological values (that is, three below and three above the reference values, if not otherwise specified), as well as to different values of the initial molecular concentrations, varied over twelve orders of magnitude with respect to their physiological values (that is, six below and six above the reference values, if not otherwise specified).

The total number of parallel simulations executed to carry out these analyses were 100 for PSA-1D over reaction constants, 200 for PSA-1D over initial concentrations and 1600 for PSA-2D.

PSA-1D of the BCC model

Reaction $\mathbf{R_{44}}$. The first PSA was performed to determine the value of the kinetic constant for the autoactivation of factor XII (reaction R_{44} in Table A.6), which corresponds to an upstream process in the intrinsic pathway. This analysis was motivated by two considerations. Firstly, by using the full parameterization given in [56], the action of the intrinsic pathway turns out to be fundamental for the BCC *in vivo*, which is in contrast to experimental observations which indicate that the extrinsic pathway is the main responsible of clot formation. Secondly, in [56] all constants values have a reference to experimental measurements, except for the constant of this reaction (which corresponds to entry 29 in Table 1 in [56]).

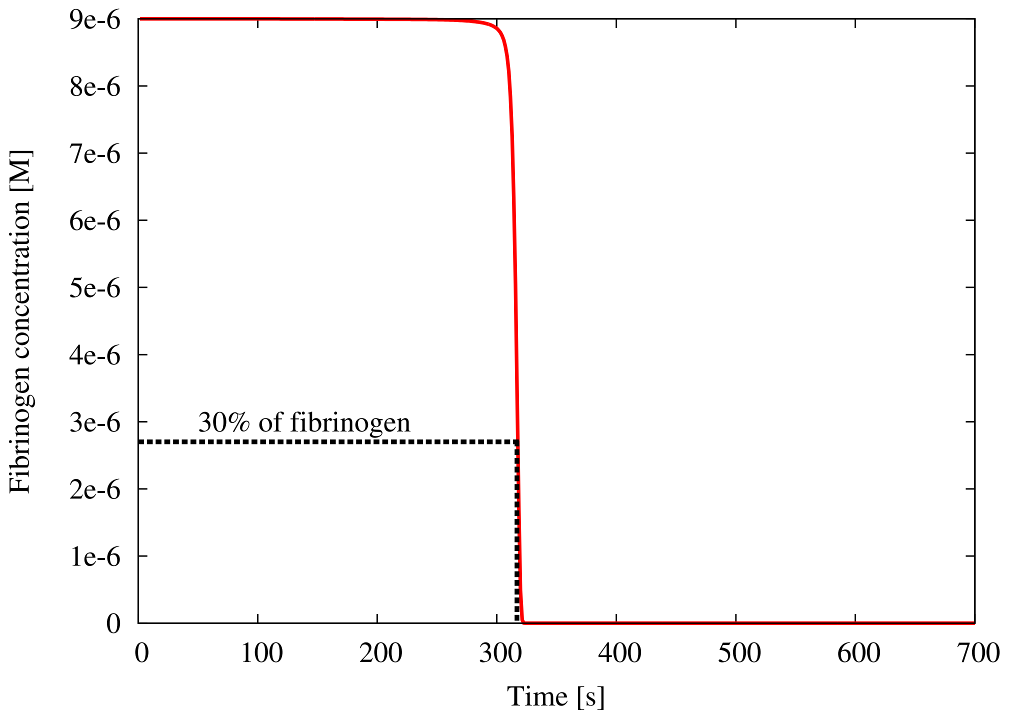


Figure 5.6: Dynamics of fibrinogen in physiological condition. The clotting time is defined as the time necessary to convert the 70% of the fibrinogen into fibrin.

Figure 5.7 shows the results of this PSA-1D, where k_{44} was varied over six orders of magnitude, considering the value given in [56] as the upper limit of the sweep interval. We can observe that, by decreasing the value of k_{44} with respect to the value considered in [56], the CT increases with respect to its reference value; however, for values of this constant lower than $1 \cdot 10^{-6}$ $\mathrm{M}^{-1}\mathrm{s}^{-1}$ the CT remains unaltered, and this can be intuitively explained by the fact that in this situation the fibrinogen is mainly activated by the extrinsic pathway.

Consequently, the value $7 \cdot 10^{-6}$ s^{-1} was assigned to the reaction constant k_{44}, achieving a CT that is comparable to the experimental observations of the BCC *in vivo*. This new value was used in all PSA discussed in what follows.

Reactions $\mathbf{R_{27}}$ and $\mathbf{R_{58}}$. The next PSA was carried out to investigate the effect of the perturbation of the kinetics of two pivotal reactions of the BCC model. Reaction R_{27}, which describes the catalytical activation of factor V by factor IIa, was chosen because it represents the main positive feedback within the common pathway; reaction R_{58}, which is involved in the activation of factor XI by binding to factor IIa, was chosen because it represents the main positive feedback in the intrinsic pathway. Moreover, preliminary PSA over all reaction constants of the BCC model given in Table A.6

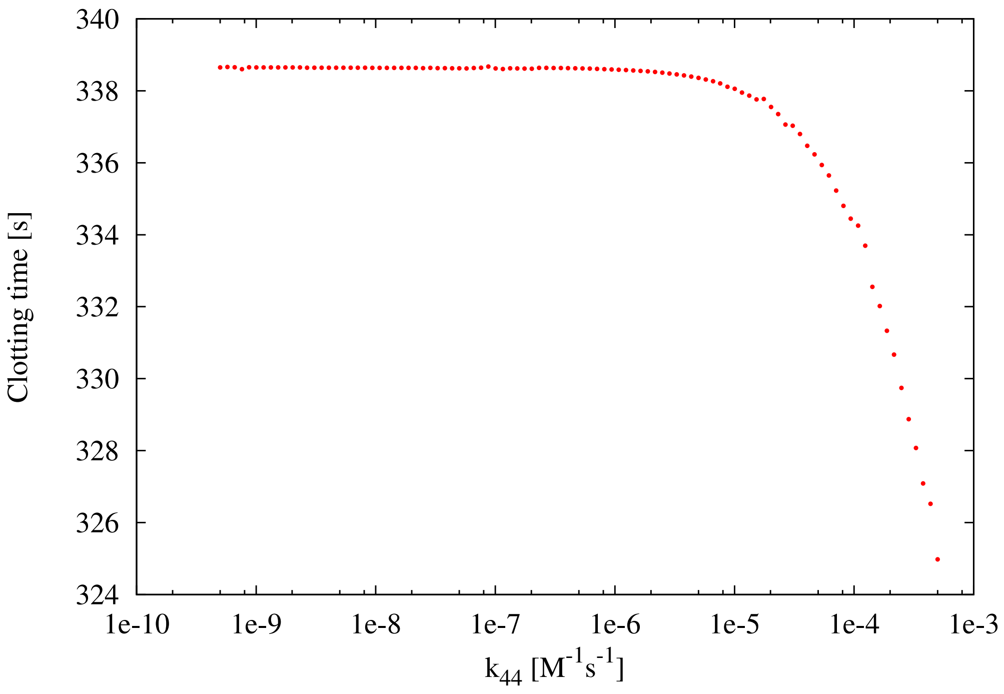


Figure 5.7: Clotting time according to PSA-1D over constant k_{44} of reaction R_{44}: fXII $\rightarrow$ fXIIa. The reference value of k_{44} used in [56] is $5{\cdot}10^{-4}$ s^{-1}, the sweep range is [$5{\cdot}10^{-10}$, $5{\cdot}10^{-4}$].

evidenced that these two reactions are among the most relevant steps of the coagulation network.

The PSA-1D over k_{27} (Figure 5.8) shows that the CT is very sensitive to the perturbation of the rate of this reaction, when the reference value of its constant (i.e., $k_{27} = 2 \cdot 10^7$ $M^{-1}s^{-1}$) is either increased or decreased; in particular, when k_{27} is very low, a plateau in CT is reached since the strength of the positive feedback exerted by factor IIa is largely reduced, a condition where the contribution of the amplification of the hemostatic stimulus (due by the common pathway) to the formation of the clot is basically not effective. On the other side, the PSA-1D over k_{58} (Figure 5.9) shows that, while a decrease of the constant with respect to its reference value (i.e., $k_{58} = 1 \cdot 10^8$ $M^{-1}s^{-1}$) does not have any substantial effect, an increase of its value leads to a progressive increase in the CT. This increase is due to the fact that factor IIa is sequestered in the formation of a complex with factor XI, and hence it is no longer available as a free component in blood to participate in other reactions, especially those reactions of the extrinsic pathway which principally lead to the clot formation *in vivo*. This behavior highlights that the intrinsic pathway has a secondary role in blood coagulation *in vivo*, compared with the extrinsic pathway, as also evidenced by various experimental observations [278].

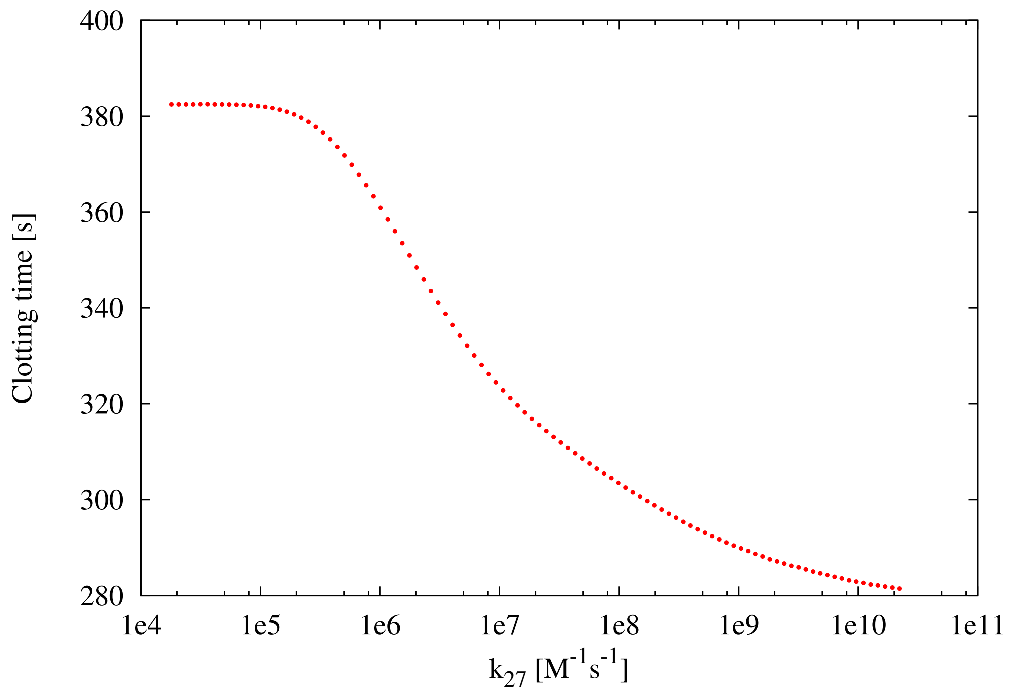


Figure 5.8: Clotting time according to PSA-1D over constant k_{27} of reaction R_{27}: fIIa + fV $\rightarrow$ fIIa + fVa. The reference value is $2{\cdot}10^7$ $M^{-1}s^{-1}$, the sweep range is [$2{\cdot}10^4$, $2{\cdot}10^{10}$].

Factors VIII, IX and II. The next set of PSA-1D was realized by varying the initial concentrations of factors VIII, IX and II. These factors were selected since both an excess or a deficiency of their concentrations lead to diseases related to blood clotting.

The PSA-1D over factor VIII (Figure 5.10) shows that increasing the initial concentration of this factor results in decreasing the CT, suggesting the possible presence of hypercoagulable states in these perturbed conditions. As a matter of fact, high levels of factor VIII cause an increased risk of deep vein thrombosis and pulmonary embolism [146]. On the other hand, individuals with less than 1% of the average concentration of factor VIII show a severe haemophilia A, characterized by higher CT (Figure 5.10), and require infusions of plasma containing the deficient factor, otherwise frequent spontaneous bleeding would occur [346]. When the concentration of this factor is between 5% and 30% of the average concentration, individuals still risk bleeding in case of trauma [346].

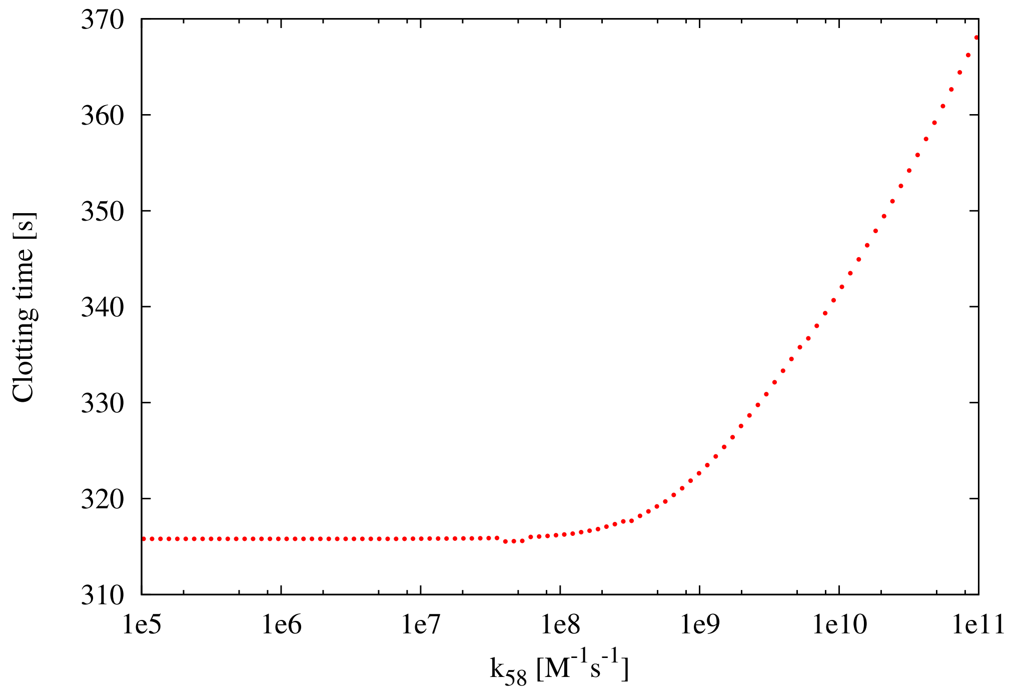


Figure 5.9: Clotting time according to PSA-1D over constant k_{58} of reaction R_{58}: fIIa + fXI → fIIa-fXI. The reference value is $1{\cdot}10^8$ $\mathrm{M}^{-1}\mathrm{s}^{-1}$, the sweep range is $[1{\cdot}10^5,\ 2{\cdot}10^{11}]$.

The PSA-1D over factor IX (Figure 5.11) shows only a slight decrease of the CT as the initial concentration of fIX increases; this is in contrast to recent studies that demonstrated how the excess of factor IX leads to an increased risk of deep vein thrombosis [338]. This result can be explained by considering that: *(i)* a high concentration of factor IX is not sufficient to bring about coagulation problems, though, when the concentration of other factors is above the average value (yet not at pathological levels), prothrombotic states can be observed; *(ii)* in this model the *average* values are considered as initial concentrations of factors, however, individuals are characterized by different (balanced) combinations of procoagulant and anticoagulant factor levels that altogether contribute to define a unique coagulation phenotype that reflects the developmental, environmental, genetic, nutritional and pharmacological influences of each individual [36]. On the contrary, the lack of factor IX cause haemophilia B, characterized by higher CT with respect to the reference value (Figure 5.11).

Furthermore, by comparing the PSA-1D of factors VIII and IX (Figures 5.10 and 5.11) it is clear that haemophilia A is more serious than haemophilia B, since the CT achieved in conditions of factor VIII deficiency is higher than the CT obtained in the case of factor IX deficiency.

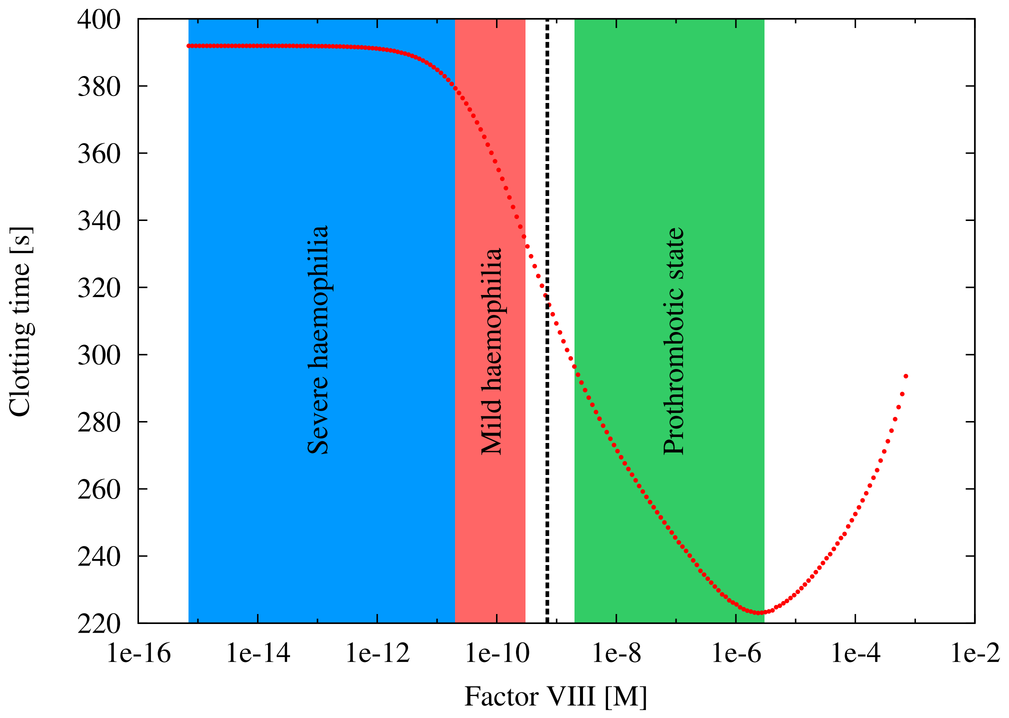


Figure 5.10: Clotting time at different initial concentrations of factor VIII. The reference value is $7{\cdot}10^{-10}$ M (dashed black line), the sweep range is $[7{\cdot}10^{-16}, 7{\cdot}10^{-4}]$. The blue area indicates a condition in which factor VIII concentration is less than 1% of its physiological concentration, corresponding to a situation of severe haemophilia; the red area indicates a condition in which factor VIII concentration is between 1% and 30% of its physiological concentration, corresponding to a situation of mild haemophilia; the green area indicates a condition in which factor VIII concentration is greater than 130% of its physiological concentration, corresponding to a situation of hypercoagulability.

In both PSA-1D over factors VIII and IX, after the initial decrease of the CT an unexpected increase of the CT was observed, as the factor concentration increases. This counterintuitive behavior arises at very high concentrations of these factors (with respect to the average physiological levels); it would be interesting to verify, by means of *ad hoc* laboratory experiments, if the model correctly describes the behavior of the BCC even in these conditions or, on the contrary, it is not predictive in these extreme situations.

The PSA-1D over factor II (Figure 5.12) shows a dramatic decrease of the CT as the initial concentration of this factor increases (with respect to the average physiological level). This behavior resembles the effects of hypercoagulability (or thrombophilia), a disease caused by mutation G20210A in the prothrombin gene [306], that causes an increase of the prothrombin level (factor II) in the blood flow, resulting in an excessive formation of the active form of this factor, thus heightening venous thrombosis risks

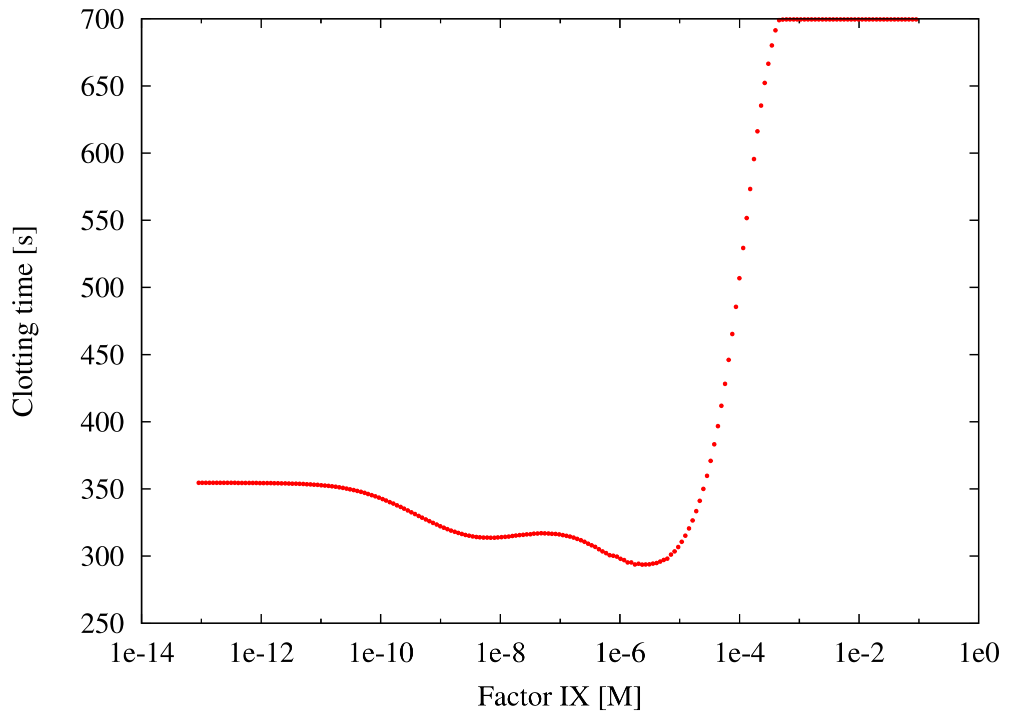


Figure 5.11: Clotting time at different initial concentrations of factor IX. The reference value is $9{\cdot}10^{-8}$ M, the sweep range is $[9{\cdot}10^{-14}, 9{\cdot}10^{-2}]$.

[201]. Hypercoagulability is usually treated with warfarin therapy, or with other anticoagulants with a similar effect. These drugs decrease the capacity of coagulation factors to become active, preventing the formation of unwanted thrombies.

On the other hand, when the initial concentration of factor II is low, the analysis results mimicked the effects of prothrombin deficiency, a rare autosomal recessive disease that causes a tendency to severe bleeding [37, 62]. As shown in Figure 5.12, a concentration equal to 10% of the physiological value of factor II (i.e., $1.4 \cdot 10^{-6}$ M) leads to clotting effects similar to severe haemophilia A.

PSA-2D of the BCC model

Reactions $\mathbf{R_{27}}$ and $\mathbf{R_{58}}$. Figure 5.13 shows the effect of the simultaneous variation of constant k_{27} and k_{58} (over the same sweep ranges previously considered in the two PSA-1D). This result remarks that reaction R_{27}, involved in the common pathway, has a stronger influence on the BCC, and that there is a synergic interplay between these two reactions. In particular, when the value of k_{27} is low and the value of k_{58} is high, the CT is higher than the values achieved when only a single constant is

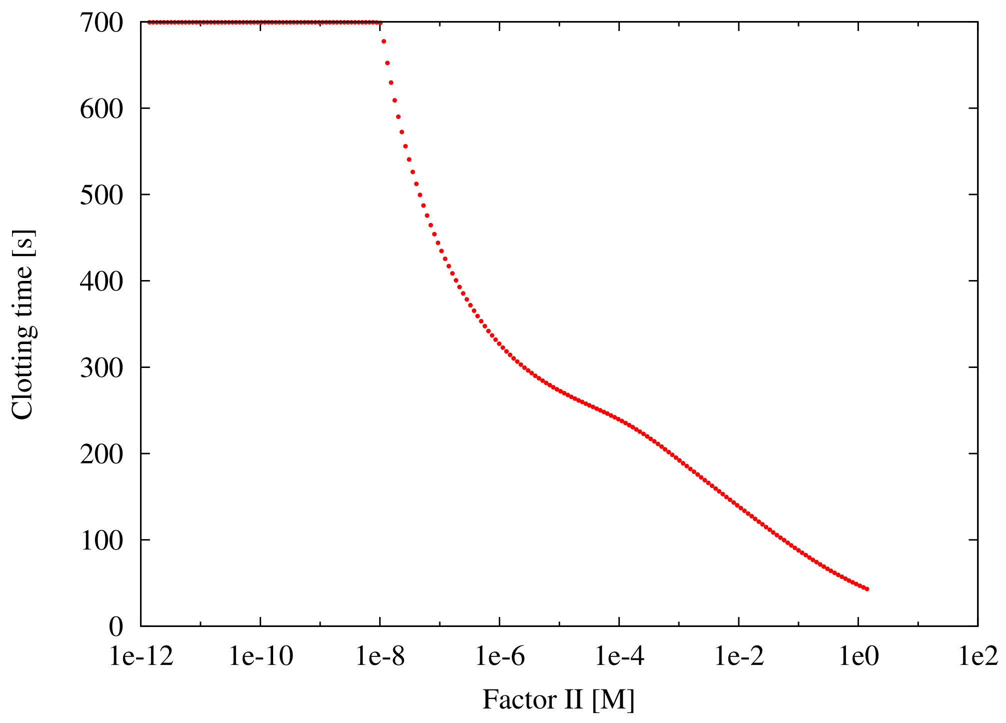


Figure 5.12: Clotting time at different initial concentrations of thrombin. The reference value is $1.40 \cdot 10^{-6}$ M, the sweep range is $[1.40 \cdot 10^{-12}, 1.40 \cdot 10^{0}]$.

changed, because in this condition both the intrinsic and the common pathways are simultaneously inhibited.

Factor VIII, factor IX and Tissue Factor. In the last two PSA-2D the variation concerned the initial concentration of factor VIII and Tissue Factor, and the initial concentration of factor IX and Tissue Factor, respectively.

The initial concentrations of factors VIII and IX were varied over four orders of magnitude, using their physiological values as upper limit for the sweep ranges; the concentration of Tissue Factor was varied over four orders of magnitude, two above and two below its reference value (see Table A.7). The rationale behind this choice is to observe how the BCC model, in conditions corresponding to different states of haemophilia (obtained by decreasing the concentrations of factors VIII and IX), behaves with different initial concentrations of the Tissue Factor, which is the upstream factor of the extrinsic pathway, i.e., the most important element of the BCC.

The results of these PSA-2D show that, with respect to the condition of haemophilia B, in the case of haemophilia A the amount of Tissue Factor (below its reference value) has a negligible influence on the CT, as indicated by the presence of a plateau in Figure 5.14; on the contrary, concerning haemophilia B, a deficiency of Tissue Factor leads

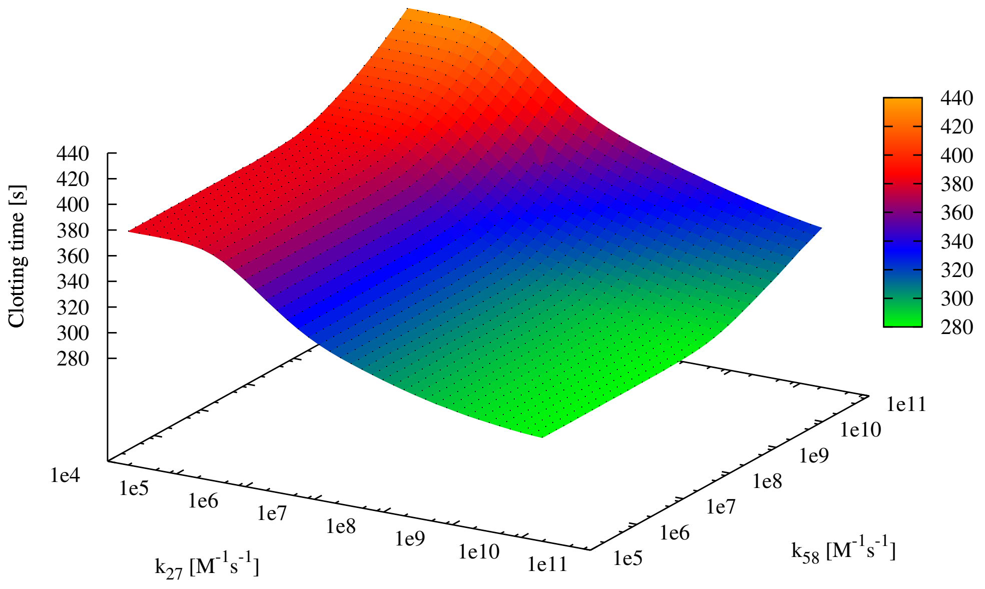


Figure 5.13: Clotting time according to PSA-2D over reaction constants k_{27} and k_{58}. The sweep ranges are $[2{\cdot}10^4, 2{\cdot}10^{10}]$ and $[1{\cdot}10^5, 2{\cdot}10^{11}]$, respectively.

to an increase of the CT, especially when factor IX is present in low concentrations (Figure 5.15).

The different results achieved in the two PSA-2D is due to the presence, in the BCC model, of a direct interaction between Tissue Factor and factor IX by means of the TF-fVIIa complex (see reactions $R_{13}, \ldots, R_{15}$ in Table A.6). Indeed, the lack of Tissue Factor directly affects the concentration of active factor IX, which results in a strong alteration of the CT with respect to physiological conditions.

CPU vs GPU performance comparison

In order to show the relevant speedup achieved by coagSODA, the comparison of the computational effort required by GPU and CPU for the simulation of the BCC model is presented here. The performances of coagSODA were compared with those obtained using the LSODA algorithm implemented in the software COPASI (version 4.8, build 35) [140], executing on the CPU the same set of simulations that were run on the GPU.

In all simulations, executed both on GPU and CPU, 100 samples were stored — uniformly distributed in the time interval considered for each simulation, i.e., [0, 700] sec — of the dynamics of all chemical species involved in the BCC model. The settings

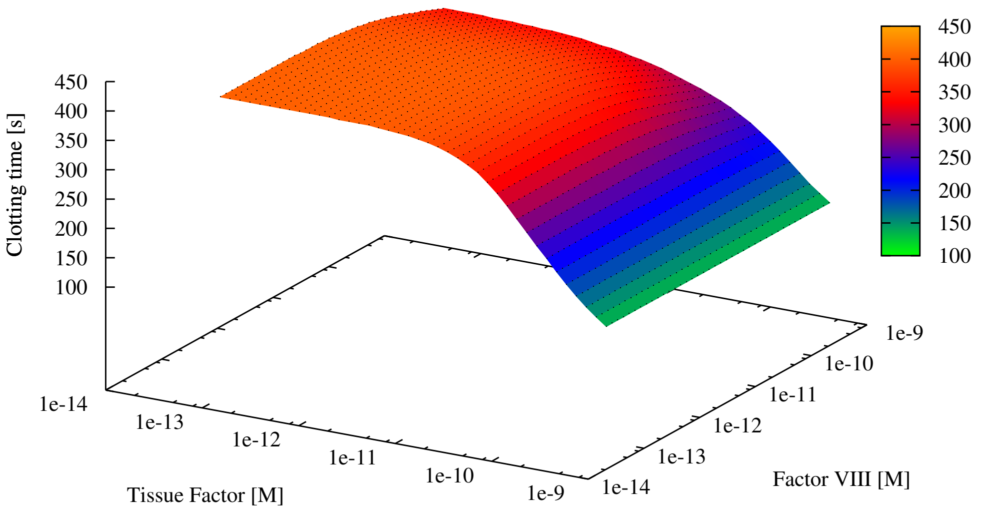


Figure 5.14: Clotting time at different initial concentrations of Tissue Factor and factor VIII. The sweep ranges are [$5{\cdot}10^{-14}$, $5{\cdot}10^{-10}$] and [$7{\cdot}10^{-14}$, $5{\cdot}10^{-10}$], respectively.

for the LSODA algorithm were the following: RET $1 \cdot 10^{-7}$, AET $1 \cdot 10^{-13}$, maximum number of internal steps set to 20000.

Benchmark details The GPU used for the simulations is a Nvidia Tesla K20c, equipped with 2496 cores organized in 13 SMXs, GPU clock 706 MHz and 5 GB of DDR5 RAM. In all tests, coagSODA was compiled and executed with version 5.5 of CUDA libraries. Even though this GPU has compute capability 3.5 and is based on the GK110 Kepler architecture, currently coagSODA does not exploit any of the new features (e.g., Dynamic Parallelism, see Section 4.1) with the exception of the new ISA encoding, which allows threads to exploit an increased number of registers (255 instead of 63), reducing register spilling into global memory and increasing performances [245]. Moreover, as described in Section 4.1, the Kepler architecture offers the possibility to reconfigure the 64KB on-chip cache, balancing between L1 cache and shared memory. Since coagSODA exploits the shared memory to reduce the latencies during the access to the concentration values of the BCC model, the following configuration was chosen: 48 KB for the shared memory, 16 KB for the L1 cache.

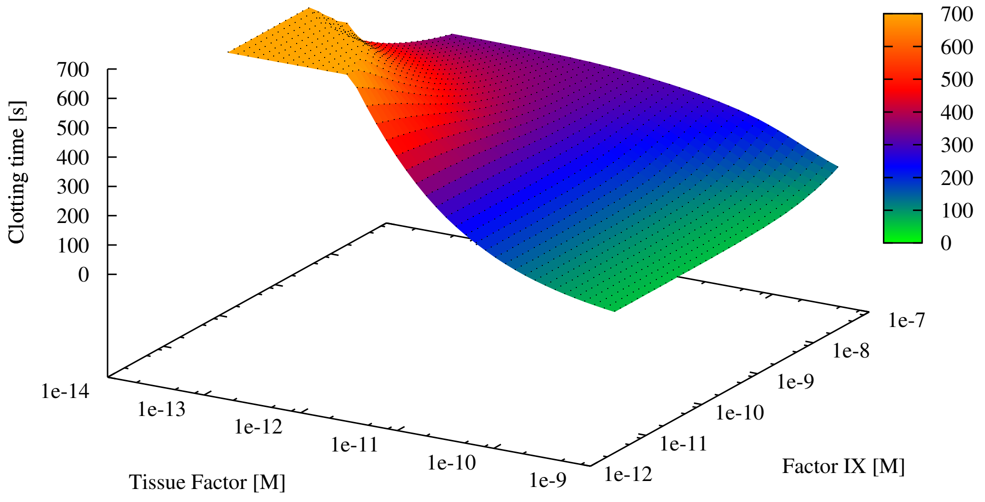


Figure 5.15: Clotting time at different initial concentrations of Tissue Factor and factor IX. The sweep ranges are [$5{\cdot}10^{-14}$, $5{\cdot}10^{-10}$] and [$9{\cdot}10^{-12}$, $9{\cdot}10^{-8}$], respectively.

The performance of the GPU was compared against a personal computer equipped with a dual-core CPU Intel Core i5, frequency 2.3 GHz, 4 GB of DDR3 RAM, running the operating system Mac OS X Lion 10.7.5.

Benchmark results Figure 5.16 shows the comparison of the running times required to run several batches of simulations, executed to carry out the PSA-1D over the reaction constant k_{27}. The choice of comparing the performances of the GPU and CPU implementations by executing batches of simulations that are related to a PSA — instead of running n independent but identical simulations (i.e., all characterized by the same parameterization of the model) — is due to the fact that these results represent a more realistic scenario in the computational analysis of biological systems, whereby it is useful to investigate large search spaces of parameters, corresponding to different perturbed conditions of the model [235]. Moreover, for the evaluation of the running time, the execution of a batch of n identical deterministic simulations would be futile. The figure clearly shows that coagSODA always performs better than the CPU counterpart. In particular, while the CPU performance increases linearly with the number of simulations, the running times are strongly reduced on the GPU: in this case, the overall running time roughly corresponds to the time required for the execution

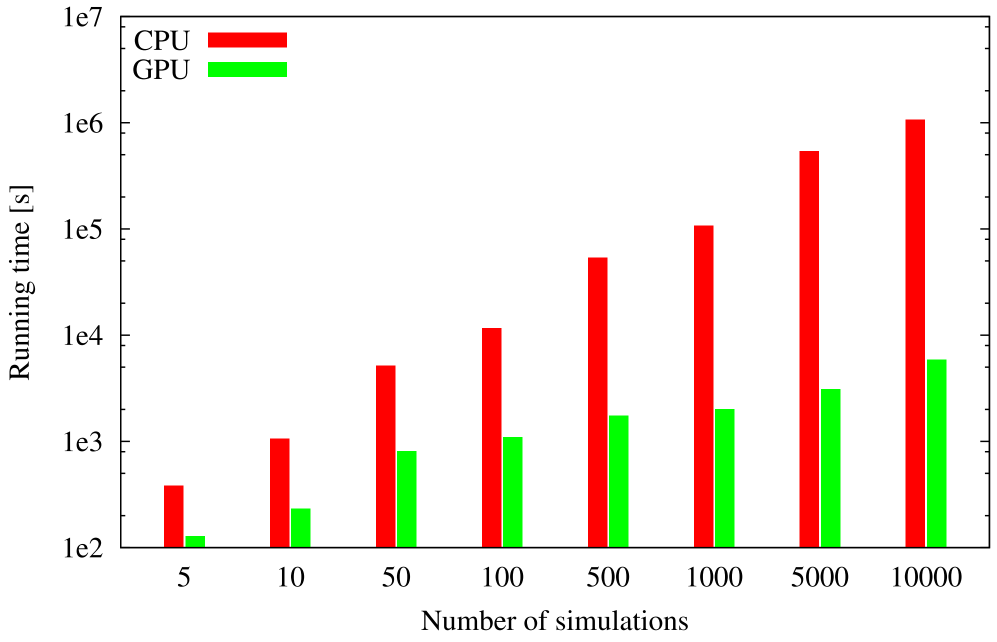


Figure 5.16: Comparison of the computational time required to execute an increasing number of simulations of the BCC model on CPU (Intel Core i5, 2.3 GHz) and GPU (Nvidia Tesla K20c, GPU clock 706 MHz). The computational time is expressed in sec. The time value related to 1000, 5000 and 10000 simulations on the CPU were estimated by regression (see also Table 5.1).

of the slowest simulations. This is due to the fact that different parameterizations may require different time steps for LSODA to converge, and the execution of a block terminates as long as all the threads it contains terminate. In turn, the execution of a kernel terminates when all blocks terminate: for this reason, a single simulation, whose dynamics requires more steps than the others, may affect the overall running time.

Table 5.1 reports the running times of all batches of simulations, along with the speedup achieved on the GPU. In particular, these results highlight that the advantage of exploiting the GPUs for the simulation of the BCC model becomes more evident as the number of simulations increases, with a 181× speedup when a PSA with 10000 different parameterizations is executed. Therefore, the GPU accelerated analysis of the BCC model with coagSODA represents a novel, relevant computational mean to investigate the behavior of this complex biological system under non-physiological conditions, and could be exploited to efficiently determine the response of the BCC to different therapeutical drugs.

Table 5.1: CPU vs GPU performance comparison.

Number of simulations	*CPU time (sec)*	*GPU time (sec)*	*Speedup*
5	380.8	127.8	2.98 ×
10	1060.4	232.9	4.56 ×
50	5170.4	805.8	6.42 ×
100	11652.8	1097.7	10.61 ×
500	53605.8	1739.4	30.81 ×
1000	107358.2*	1998.2	53.73 ×
5000	536350.7*	3096.0	173.2 ×
10000	1072497.5*	5895.1	181.9 ×

* Values estimated by regression

5.3 Stochastic biochemical simulation: cuTauLeaping

This section presents the development of a GPU-powered stochastic simulator of biochemical systems based on the tau-leaping algorithm, named cuTauLeaping, and its application to perform parallel stochastic simulations in a massively parallel way, by running multiple independent simulations as parallel CUDA threads. The peculiar GPU-oriented design of cuTauLeaping and its data structures, memory allocation strategies and advanced functions exploited on the Fermi architecture, are presented in detail.

5.3.1 GPU implementation of cuTauLeaping

In cuTauLeaping, the workflow of the traditional tau-leaping algorithm — described in Section 2.3.2 — is partitioned in different phases, which altogether allow a better exploitation of the parallel architecture of the GPU than a monolithic implementation. The rationale behind this choice is that the resources on each SMX are limited, thus they would be quickly consumed by the data structures employed by tau-leaping, causing a low occupancy of the GPU that would then result in worse performances. Moreover, since tau-leaping embeds the potential execution of SSA simulation steps, a "fat" kernel responsible for both simulation algorithms would not be convenient, because of the following issues:

- when the largest permissible time step τ for non-critical reactions is very low, it is faster to forgo tau-leaping and to execute an arbitrary number of SSA steps (see Step 3 in the modified Poisson tau-leaping algorithm presented in [46]);
- SSA is simpler and requires fewer resources than tau-leaping (the only thing the two algorithms share is the vector of propensity functions).

Therefore, the partitioning of tau-leaping workflow in different phases allows a faster execution of the simulations, thanks to the reduced memory footprint, which yields a higher level of parallelism.

The four phases that constitute cuTauLeaping, schematized in Figure 5.17, are:

- **phase** `P1`: each thread i, where $i = 1, \dots, U$, and $U \in \mathbb{N}$ is specified by the user, determines a tentative value for the length of the time step τ for the non-critical reactions, by using Equation 2.6;
- **phase** `P2`: all threads where the length of the time step is such that $\tau \geq \frac{10}{a_0(\mathbf{x})}$ execute a tau-leaping step;
- **phase** `P3`: the remaining threads execute a fixed number of SSA steps (100 is the default setting);
- **phase** `P4`: check the termination criterion of the simulation in all threads (cuTauLeaping termination).

Each thread proceeds by applying tau-leaping or SSA steps, which are mutually exclusive, according to the value of a vector $\mathbf{Q} \in \{-1, 0, 1\}^U$, where for each $i = 1, \dots U$ the element $q_i \in \mathbf{Q}$ is set to 0 if the i-th thread must execute SSA, 1 if the i-th thread must execute tau-leaping, while the value -1 corresponds to the signal of terminated simulation.

In cuTauLeaping the first two phases are implemented in a single kernel, so that the tau-leaping step can be executed right after the calculation of the τ value, without the need for a global memory write (e.g., to update $\mathbf{Q}$) or a recalculation of the propensity functions and τ (that could be required, in such a case), which would reduce the performances. In particular, during the first two phases, after the computation of the putative τ value for non-critical reactions, a second putative time step value related to critical reactions is calculated, and the smallest one is used in the current tau-leaping

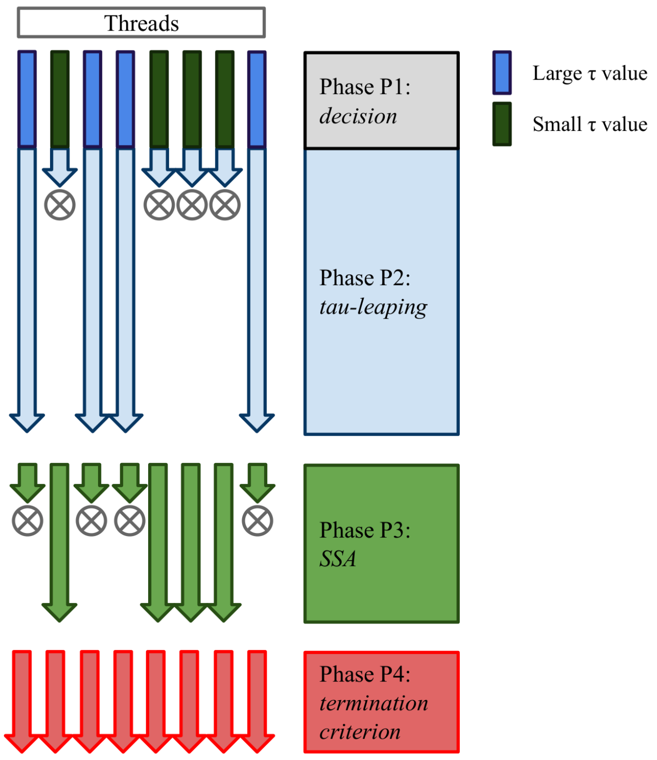


Figure 5.17: Simplified scheme of cuTauLeaping workflow: in phase `P1` each thread calculates the value τ for the simulation step; in phase `P2`, the threads whose τ is “large” perform a tau-leaping step (by executing a set of non-critical reactions and (possibly) one critical reaction); the remaining threads perform a fixed number of SSA steps (where one reaction is executed at each step) during phase `P3`. The phases are iterated until all threads have reached $t_{\texttt{MAX}}$, a termination criterion verified during phase `P4`.

step. If the first putative τ value is used, then only non-critical reactions sampled from the Poisson distributions are applied; otherwise, besides non-critical reactions, also one critical reaction is selected and applied (as described in [46]).

The four phases are implemented in the following kernels, which are executed in a sequential manner by each thread i:

- kernel `P1-P2`: if $q_i = -1$, then terminate the kernel; otherwise, calculate the τ value for non-critical reactions.
  If $\tau < \frac{10}{a_0(\mathbf{x})}$, then $q_i \leftarrow 0$ and terminate the kernel; else $q_i \leftarrow 1$ and execute a tau-leaping step updating the system state $\mathbf{x}$ (according to Equation 2.5, by executing a set of non-critical reactions and, possibly, one critical reaction) and the global simulation time (by setting $t \leftarrow t + \tau$). If $t > t_{\texttt{MAX}}$, then $q_i \leftarrow -1$ and terminate the kernel;

- kernel P3: if $q_i \neq 0$, then terminate the kernel; otherwise, perform the SSA steps (by executing a single reaction at each step), and update the system state $\mathbf{x}$ (according to Equation 2.5) and the global simulation time t (by setting, at each SSA step, $t \leftarrow t + \tau$). If $t > t_{\texttt{MAX}}$ set $q_i \leftarrow -1$ and terminate the kernel;

- kernel P4: if $q_i = -1$ (for all threads), then terminate cuTauLeaping; else go back to kernel P1-P2.

Figure 5.18 shows the pseudocode of the host side procedure devoted to invoke the CUDA kernels; Figures 5.19, 5.20, 5.21, present the pseudocodes of kernels P1-P2, P3, P4, respectively.

Algorithm 1 Host-side pseudocode of cuTauLeaping.

```
1: procedure cuTauLeaping($\mathbf{MA}[\,][\,], \mathbf{MB}[\,][\,], \mathbf{x}_0[\,][\,], \mathbf{c}[\,][\,], \mathbf{I}[\,], \mathbf{E}[\,], U$)
2:     $\mathbf{A}, \mathbf{V}, \mathbf{V}^{\mathrm{T}}, \bar{\mathbf{V}}, \mathbf{H}, \mathbf{H}_{\mathbf{type}}$ ← CalculateDataStructures($\mathbf{MA}, \mathbf{MB}, \mathbf{x_0}, \mathbf{c}$)
3:     LoadDataOnGPU( $\mathbf{A}, \mathbf{V}, \mathbf{V}^{\mathrm{T}}, \bar{\mathbf{V}}, \mathbf{H}, \mathbf{H}_{\mathbf{type}}, \mathbf{x}_0, \mathbf{c}$ )
4:     AllocateDataOnGPU( $\mathbf{t}, \mathbf{x}, \mathbf{O}, \mathbf{E}, \mathbf{Q}$ )
5:     gridSize, blockSize ← DistributeWorkload($U$)
6:     repeat
7:         Kernel$_{\mathbf{P1-P2}}$ <<<gridSize,blockSize>>>
8:             ( $\mathbf{A}, \mathbf{V}, \mathbf{V}^{\mathrm{T}}, \bar{\mathbf{V}}, \mathbf{H}, \mathbf{H}_{\mathbf{type}}, \mathbf{x}, \mathbf{c}, \mathbf{I}, \mathbf{E}, \mathbf{O}, \mathbf{Q}, \mathbf{t}$ )
9:         Kernel$_{\mathbf{P3}}$ <<<gridSize,blockSize>>>
10:            ( $\mathbf{A}, \mathbf{V}, \mathbf{x}, \mathbf{c}, \mathbf{I}, \mathbf{E}, \mathbf{O}, \mathbf{Q}, \mathbf{t}$ )
11:        TerminSimulations ← Kernel$_{\mathbf{P4}}$ <<<gridSize,blockSize>>>( $\mathbf{Q}$ )
12:    until TerminSimulations = $U$
13: end procedure
```

Figure 5.18: Host-side pseudocode of cuTauLeaping. As a first step, the stoichiometric information of the reactions is exploited to pre-calculate the data structures needed by the algorithm; all matrices are flattened during this process. Then, once the support memory areas are allocated (e.g., the chunk of global memory where the system dynamics will be stored), the four phases of cuTauLeaping begin and are repeated until all simulations are completed.

Kernels are iteratively repeated until $t \geq t_{\texttt{MAX}}$ for all threads. This termination criterion is efficiently verified by kernel P4 that exploits two advanced CUDA functionalities introduced with the Fermi architecture: *synchronizations with predicate evaluation* and *atomic functions*. Synchronization functions are generally used to coordinate the communication between threads, but CUDA allows to exploit these functions to evaluate a predicate for all threads in a block; atomic functions allow to perform read-modify-write operations without any interference from any other thread, therefore avoiding the race condition. A combination of these functionalities allows to determine whether all threads have terminated their execution (i.e., the predicate is $q_i = -1$, for all $i = 1, \ldots, U$). In addition, since both functionalities are hardware-accelerated,

Algorithm 2 Phases P1, P2 of cuTauLeaping: subdivision of threads according to the τ value and execution of the tau-leaping steps.

```
procedure KERNEL$_{\text{P1-P2}}$($\mathbf{A}[\,], \mathbf{V}[\,], \mathbf{V}^{\text{T}}[\,], \bar{\mathbf{V}}[\,]$, global_x[ ][ ], global_c[ ][ ],
        $\mathbf{I}[\,], \mathbf{H}[\,], \mathbf{H}_{\text{type}}[\,], \mathbf{E}[\,], \mathbf{O}[\,][\,], \mathbf{Q}[\,], \mathbf{t}[\,]$)
    $tid \leftarrow$ getGlobalId()                         ▷ Global-level thread identifier
    $sid \leftarrow$ getLocalId()                          ▷ Block-level memory identifier
    $\mathbf{x}[sid] \leftarrow$ global_x[$tid$]
    $\mathbf{c}[sid] \leftarrow$ global_c[$tid$]
    if $\mathbf{Q}[tid] = -1$ then return                 ▷ Signal of terminated simulation
    end if
    $\mathbf{a} \leftarrow$ UpdatePropensities( $\mathbf{x}[sid], \mathbf{c}[tid]$ )
    $a_0 \leftarrow$Sum($\mathbf{a}$)
    if $a_0 = 0$ then
        for $i \leftarrow \mathbf{F}[tid] \dots \iota$ do
            $\mathbf{O}[tid][i] \leftarrow$ GetSpecies($\mathbf{x}[sid]$, $\mathbf{E}$)
        end for
        $\mathbf{Q}[tid] \leftarrow -1$
        return                                          ▷ No reactions can be applied
    end if
    $\chi \leftarrow$ DetermineCriticalReactions( $\mathbf{A}, \mathbf{x}[sid]$ )
    $\mu, \sigma \leftarrow$ CalculateMuSigma( $\mathbf{x}[sid], \mathbf{H}, \mathbf{H}_{\text{type}}, \mathbf{a}[sid], \chi[sid]$ )
    $\tau_1 \leftarrow$ CalculateTau($\mu, \sigma$)
    if $\tau_1 < 10./a_0$ then
        $\mathbf{Q}[tid] \leftarrow 0$
        return                                          ▷ SSA steps are more efficient
    else
        $\mathbf{Q}[tid] \leftarrow 1$                             ▷ tau-leaping will be performed
    end if
    $\mathbf{K} \leftarrow [\,], a_{0_c} \leftarrow 0$
    for all $j$ in $\chi$ do
        $a_{0_c} \leftarrow a_{0_c} + \mathbf{a}[j]$           ▷ Sum of the propensities of critical reactions
    end for
    if $a_{0_c} > 0$ then $\tau_2 \leftarrow (1./a_{0_c}) * \ln(1./\rho_1)$
    end if
    $\tau \leftarrow \min\{\tau_1, \tau_2\}$
    if $\tau_1 > \tau_2$ then
        $j \leftarrow$ SingleCriticalReaction( $\chi[tid], \mathbf{a}[tid]$ )
        $\mathbf{K}[j] \leftarrow 1$
    end if
    repeat
        for all $j$ not in $\chi$ do
            $\mathbf{K}[j] \leftarrow$ Poisson( $\tau, \mathbf{a}[sid][j]$ )
        end for
        $\mathbf{x}' \leftarrow$ TentativeUpdatedState($\mathbf{x}[sid], \mathbf{K}[sid], \mathbf{V}$)
        if ValidState( $\mathbf{x}'$ ) then break
        end if
        $\tau \leftarrow \tau/2$                                    ▷ Negative species detected
    until False
    $\mathbf{t}[tid] \leftarrow \mathbf{t}[tid] + \tau$
    while $\mathbf{t}[tid] \geq \mathbf{I}[\mathbf{F}[tid]]$ do
        SaveInterpolatedDynamics($\mathbf{x}, \mathbf{x}', \mathbf{O}[tid], \mathbf{F}[tid]$)
        $\mathbf{F}[tid]++$
        if $\mathbf{F}[tid] = \iota$ then
            $\mathbf{Q}[tid] \leftarrow -1$                    ▷ No more samples: simulation over
        end if
    end while
    global_x $\leftarrow \mathbf{x}'$
end procedure
```

Figure 5.19: Device-side pseudocode of kernel P1-P2, implementing the subdivision of threads according to the τ value and the execution of a tau-leaping step.

Algorithm 3 Phase P3 of cuTauLeaping: potential execution of the SSA steps.

```
procedure $\textsc{Kernel}_{\texttt{P3}}(\mathbf{A}[\,], \mathbf{V}[\,], \mathbf{global\_x}[\,][\,], \mathbf{global\_c}[\,][\,], \mathbf{I}[\,],$
          $\mathbf{E}[\,], \mathbf{O}[\,][\,], \mathbf{Q}[\,], \mathbf{t}[\,])$
    $tid \leftarrow$ getGlobalId()                          ▷ Global-level thread identifier
    $sid \leftarrow$ getLocalId()                           ▷ Block-level memory identifier
    if $\mathbf{Q}[tid] \neq 0$ then return                 ▷ SSA steps not needed
    end if
    $\mathbf{x}[sid] \leftarrow \mathbf{global\_x}[tid]$
    $\mathbf{c}[sid] \leftarrow \mathbf{global\_c}[tid]$
    for $i$ in $0 \dots 100$ do
        $\mathbf{a}[sid] \leftarrow$ UpdatePropensities( $\mathbf{x}[sid], \mathbf{c}[sid]$ )
        $a_0 \leftarrow$Sum($\mathbf{a}[sid]$)
        if $a_0 = 0$ then
            for $i \leftarrow \mathbf{E}[tid] \dots \iota$ do
                $\mathbf{O}[tid][i] \leftarrow \mathbf{x}[tid]$
            end for
            $\mathbf{Q}[tid] \leftarrow -1$
            return                                          ▷ No reactions can be applied
        end if
        $\tau \leftarrow (1./a_0) * \ln(1./\rho_1)$
        $j \leftarrow$ SingleCriticalReaction( $\chi[sid], \mathbf{a}[sid]$ )
        $\mathbf{T}[tid] \leftarrow \mathbf{T}[tid] + \tau$
        if $\mathbf{T}[tid] \geq \mathbf{I}[\mathbf{F}[tid]]$ then
            SaveDynamics($\mathbf{x}[sid], \mathbf{O}[tid], \mathbf{E}[tid]$)
            $\mathbf{F}[tid]++$
            if $\mathbf{F}[tid] = \iota$ then
                $\mathbf{Q}[tid] \leftarrow -1$                 ▷ No more samples: simulation over
            end if
        end if
        $\mathbf{x} \leftarrow$ UpdateState( $\mathbf{x}, j$ )
    end for
    $\mathbf{global\_x} \leftarrow \mathbf{x}$
end procedure
```

Figure 5.20: Device-side pseudocode of kernel P3 in cuTauLeaping, implementing the execution of the SSA steps. The kernel starts by loading the vectors **global_x** and **global_c** — which correspond to the current state of the system and to the values of stochastic constants, respectively — from the global memory areas that contain these data for all threads. Since these information are frequently accessed, they are immediately copied into the faster shared memory as vectors **x** and **c**, respectively. Then, it performs a fixed number of SSA steps (100 in the default setting), where a single reaction is executed at each step.

the resulting computational complexity is $O(\#\text{blocks})$, making them more efficient than other equivalent methodologies, e.g., parallel reduction [352], whose complexity is $O(\log U)$ (note that, in general, $\#\text{blocks} < \log U$).

In order to further improve the performance of the simulation execution, it is better not to code the stoichiometric information by means of matrices. In cuTauLeaping, stoichiometric matrices $\mathbf{MA}, \mathbf{MV}$, $\mathbf{MV}^{\mathsf{T}}$ and $\overline{\mathbf{MV}}$ of the RBM (see Section 2.2.1) — which are typically sparse matrices — are flattened by packing their non-zero elements into arrays of CUDA vector types, named *uchar*4, whose components are accessed by means of *variable.x*, *.y*, *.z*, and *.w*; the vectors corresponding to these matrices are

Algorithm 4 Phase P4 of cuTauLeaping: count of terminated simulations.

```
1: procedure KERNEL_P4(Q[ ], $Termination$)
2:     $tid \leftarrow$ getGlobalId()                        ▷ Global-level thread identifier
3:     $sid \leftarrow$ getLocalId()                         ▷ Block-level memory identifier
4:     if $tid = 0$ then
5:         $Termination \leftarrow 0$
6:     end if
7:     SynchronizeThreads()
8:     $BlockResult \leftarrow$ CountTerminatedThreads( $\mathbf{Q}$ )
9:     if $gid = 0$ then
10:        AtomicAdd($BlockResult, Termination$)
11:    end if
12: end procedure
```

Figure 5.21: Device-side pseudocode of kernel P4 in cuTauLeaping, implementing the verification of the termination of all simulations. The verification is performed by means of CUDA's hardware accelerated synchronization and counting features, which allow to count the threads of a block which satisfy a specific predicate. By exploiting CUDA's atomic functions, the total number of threads which satisfy the predicate $q_i = -1$ is accumulated: if it is equal to the number of threads, the execution of all parallel simulations is completed.

named $\mathbf{A}, \mathbf{V}, \mathbf{V}^{\mathrm{T}}, \bar{\mathbf{V}}$, respectively. Since both $\mathbf{V}^{\mathrm{T}}$ and $\bar{\mathbf{V}}$ vectors can assume negative numbers, an offset is used to store their $.z$ values as *unsigned chars* and re-calculate the correct negative values on the GPU. An example of this implementation strategy is shown in Figure 5.22 which schematizes, for the MM model (see Appendix A), the conversion of the matrix **MA** into the corresponding flattened representation **A**.

By using this strategy, the complexity of the calculations needed for both SSA and tau-leaping decreases from $\Theta(M \cdot N)$ to $\Theta(Z)$, where M is the number of reactions, N is the number of species and $Z = |\{\alpha_{ji} \in \mathbf{MA} \mid \alpha_{ji} \neq 0\}|$ is the number of non-zero entries in **MA**. For each non-zero entry, the $.x$ and $.y$ components store the corresponding row and column indices of **MA**, respectively; the $.z$ component is used to store the stoichiometric value. Note that, even though the $.w$ component is left unused, it is more efficient to employ the $uchar4$ vector type rather than $uchar3$, because the former is 4-aligned and takes a single instruction to fetch the whole entry, while the latter is 1-aligned, and would require three memory operations to read each entry of the flattened vector. It is worth noting that the use of an *unsigned char* data type implies that cuTauLeaping could deal with models with up to 256 reactions and molecular species; for larger systems, data types with greater size must be exploited. Anyway, the maximum size of a model is also limited by the shared memory available on the GPU; a detailed analysis of this issue is provided in the Section 5.3.3.

cuTauLeaping is also optimized for what concerns the calculation of $\mu_i(\mathbf{x})$ and $\sigma_i^2(\mathbf{x})$ values (Equation 2.7). These values are related to each species S_i, and represent

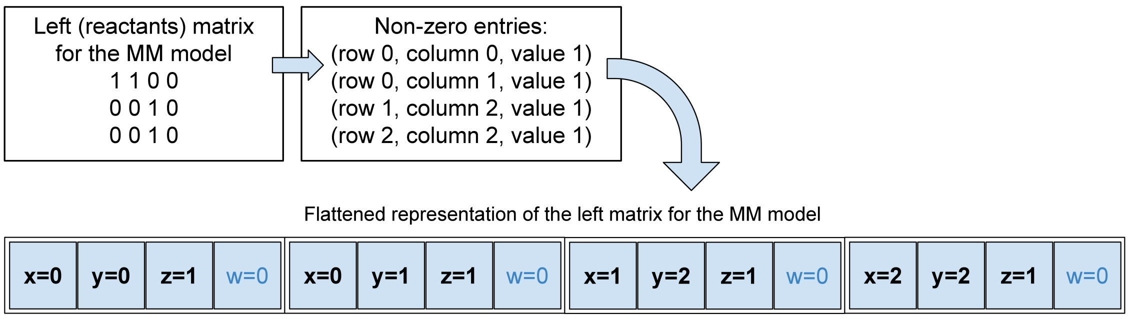

Left (reactants) matrix
for the MM model
1 1 0 0
0 0 1 0
0 0 1 0
Non-zero entries:
(row 0, column 0, value 1)
(row 0, column 1, value 1)
(row 1, column 2, value 1)
(row 2, column 2, value 1)
Flattened representation of the left matrix for the MM model
x=0 y=0 z=1 w=0
x=0 y=1 z=1 w=0
x=1 y=2 z=1 w=0
x=2 y=2 z=1 w=0

Figure 5.22: Schematization of the flattened representation of the stoichiometric information in cuTauLeaping. Each stoichiometric matrix can be pre-processed to identify its non-zero values and discard the remaining ones, thus reducing the number of reading operations required by the stochastic algorithms. A strategy to reduce the size of these matrices consists in flattening each matrix as a vector of triples (r, c, v), where r is the row index, c is the column index and v is the non-zero value in (r, c). Both r and c indices are 0-based and triples are stored using vectors of CUDA's $uchar4$ data types, that have the advantage of requiring a single instruction to fetch an entry. The top part of this figure shows the values appearing in the 3×4 stoichiometric matrix of reactant species of the MM model, which consists of 3 reactions over 4 molecular species (see Appendix A). The bottom part of the figure shows the corresponding $uchar4$ vector.

an estimate of the change of the propensity functions, based on all possible reactions in which the species S_i is involved. For this reason, the flattened representation of the matrix $\mathbf{MV}$ cannot be exploited; therefore, to obtain an efficient calculation of these values, a flattened transposed stoichiometric matrix $\mathbf{V}^{\mathrm{T}}$ is introduced.

In order to further increase the performances of cuTauLeaping, the CUDA code was optimized to better exploit the GPU architecture. The first optimization consists in keeping the register pressure low, in order to avoid the *register spilling* into global memory and to increase the occupancy of the GPU. This is achieved by partitioning the tau-leaping algorithm into multiple kernels, allowing a strong reduction of the consumption of hardware resources (i.e., the register pressure). This CUDA optimization technique, known as *branch splitting*, was shown to achieve a relevant gain of performances [48].

Another typical optimization of CUDA is to ensure coalesced access to data, i.e., an aligned and sequential organization of the memory, for all data structures that are updated by each thread. In this implementation coalescence is granted to all data structures that are private to each thread, that is, the system state $\mathbf{x}$, the stochastic constants $\mathbf{c}$, and so on. However, there is some shared information that is not inherently coalesced:

- the stoichiometric flattened vectors $\mathbf{A}$ and $\mathbf{V}$ (used by kernel `P1-P2` and kernel `P3`), $\mathbf{V}^{\mathsf{T}}$ and $\bar{\mathbf{V}}$ (used by kernel `P1-P2`);
- the vector $\mathbf{H} \in \mathbb{N}^N$ (used by kernel `P1-P2`), containing the highest order of all reactions in which each molecular species appears as a reactant;
- the vector $\mathbf{H_{type}} \in \mathbb{N}^N$ (used by kernel `P1-P2`), containing the information about the maximum number of reactant molecules involved in the highest-order reactions.

The data structures used to store the stoichiometric information $(\mathbf{A}, \mathbf{V}, \mathbf{V}^{\mathsf{T}}, \bar{\mathbf{V}})$ are not modified during the simulation and are common to all threads, and can be conveniently loaded into the constant memory. This peculiar CUDA memory is immutable and cached, so that the uncoalesced access pattern does not have any impact on the performances.

Note that, in contrast to other GPU implementations of tau-leaping [165], cuTauLeaping exploits the vector $\mathbf{A}$ to correctly evaluate the propensity functions of the reactions whose reactant species appear also as products in the same reaction: in cases like this, the net balance of the consumed and produced chemicals that is stored in vector $\mathbf{V}$ would not carry sufficient information to distinguish between reactants and products. For instance, given $\mathbf{MV}_j = (-1, -1, 1)$, it is impossible to establish whether this state change vector corresponds to a reaction of the form $S_1 + S_2 \to S_3$ or $2S_1 + 2S_2 \to S_1 + S_2 + S_3$; on the contrary, the information stored in $\mathbf{A}$ allows to discriminate between the two cases.

$\mathbf{H}$ and $\mathbf{H_{type}}$ vectors are necessary to evaluate the length of the step τ (Equation 2.6), which is determined at each step according to the current state of the system; τ is then exploited to update the simulation time of each thread, denoted by $\mathbf{t} \in \mathbb{R}^U$. Both $\mathbf{H}$ and $\mathbf{H_{type}}$ vectors, anyway, can be calculated offline by preprocessing the stoichiometric matrices $\mathbf{MA}$ and $\mathbf{MB}$ while they are loaded.

In addition, both kernels `P1-P2` and `P3` exploit the following three vectors:

- $\mathbf{x} \in \mathbb{N}^N$ is the current state of the system;
- $\mathbf{a} \in \mathbb{R}^M$ contains the values of the propensity functions of the reactions;
- $\mathbf{c} \in \mathbb{R}^M$ contains the values of the stochastic constants.

Kernel `P1-P2` exploits four additional vectors:

- $\mathbf{x}' \in \mathbb{Z}^N$ is the putative state of the system (note that the elements of this vector might assume negative values);

- $\boldsymbol{\chi} \in \{0, 1\}^M$ contains the information about critical reactions, stored as 1s, and non-critical reactions, stored as 0s;
- $\mathbf{G} \in \mathbb{R}^N$ contains the auxiliary values used to calculate the length of the time step satisfying the leap condition, which are computed by using the vectors $\mathbf{x}$, $\mathbf{H}$ and $\mathbf{H_{type}}$ (see details in [46]);
- $\mathbf{K} \in \mathbb{N}^M$ contains the samples of the Poisson distributions corresponding to the number of times each reaction will be fired in the current step.

These vectors are coalesced, but frequently exploited by tau-leaping and SSA. In order to minimize the latencies due to the frequent access to the global memory, for each thread the vectors $\mathbf{x}$, $\mathbf{x}'$, $\mathbf{a}$, $\mathbf{c}$, $\boldsymbol{\chi}$ and $\mathbf{K}$ are allocated into the shared memory. Being an on-chip memory, latencies of the shared memory are about two orders of magnitude lower than that of the global memory; the use of shared memory allows a reduction of the global bandwidth usage [290] and provides a relevant performance boost. In contrast, $\mathbf{G}$ is memorized into the global memory, since its values are used only twice during the simulation step to determine the τ value. Since cuTauLeaping was specifically designed to be embedded into other applications — in particular, the computational tools for PE and RE described in this thesis which rely on the execution of a large number of simulations — the stochastic constants are also copied into the shared memory vector $\mathbf{c}$.

To obtain a more efficient implementation, an additional strategy consisted in restructuring the tau-leaping algorithm in order to avoid the conditional branches as much as possible. In cuTauLeaping, branches were removed by unrolling loops and by allowing redundant calculations in favor of a uniform control flow.

There are two more relevant facets of cuTauLeaping implementation: the storage of the simulated dynamics and the "feed" of molecular species amounts. The storage of the entire temporal evolution of all species, associated to each thread in the GPU, cannot be realized, since it is impossible to determine *a priori* how many steps each simulation will take. Indeed, whenever a kernel is launched, the required amount of memory must be statically pre-allocated from the host; it is therefore fundamental to set the number of time instants in which the dynamics is sampled, before each simulation starts. Moreover, the naïve storage of all molecular species, even in the case of a few sampled time instants, may be unfeasible. For instance, in the case of the RBM model of the Ras/cAMP/PKA pathway (see Appendix A), storing the dynamics of 2^{12} simulations, considering 1000 samples stored as single-precision floating point values, would require 4(bytes)$\times$1000(samples)$\times$33(species)$\times 2^{12}$(simulations) $\simeq$ 0.5GB, which is very close to the memory amount of an average GPU. Therefore, cuTauLeaping makes use of four additional global memory vectors:

- $\mathbf{I} \in \mathbb{R}^{\iota}$ contains the $\iota \in \mathbb{N}$ time instants in which the temporal evolution of the system is stored;
- $\mathbf{E} \in \mathbb{N}^{\kappa}$ contains the indices of the $\kappa \in \mathbb{N}$ molecular species whose amounts are stored (for some $\kappa \leq N$);
- $\mathbf{O} \in \mathbb{N}^{\kappa \times \iota \times U}$ contains the U simulated dynamics of the molecular species in $\mathbf{E}$, sampled during the time instants in $\mathbf{I}$;
- $\mathbf{F} \in \mathbb{N}^{U}$ stores, for each thread, the pointer to the next time instant in $\mathbf{O}$, i.e., when the u-th simulation reaches the i-th sampling time instant, $\mathbf{F}_u$ is set to $i + 1$.

Finally, in order to fully exploit the cache, also the values of ι, κ, M, N, θ_c, ϵ, $t_{\texttt{MAX}}$ and the size of the flattened vectors $\mathbf{A}, \mathbf{V}, \mathbf{V}^{\mathrm{T}}$ and $\bar{\mathbf{V}}$, are stored as constants into the constant memory of the GPU.

5.3.2 Results

The computational performances of cuTauLeaping were compared with the CPU implementation of tau-leaping provided in the software COPASI [140]. To this aim, four stochastic RBMs of biological systems were used as a benchmark, namely, the three models already mentioned in Section 5.1.2 and the Schlögl model [342, 354]. The definition of each model, as well as the values of the initial molecular amounts and of the stochastic constants used to run simulations, are given in Appendix A. In addition, to analyze the influence of the size of the model (i.e., number of reactions and molecular species) on the performances of cuTauLeaping, several tests were executed on randomly generated synthetic models characterized by different size and various parameterizations.

The advantages of using cuTauLeaping to investigate the effects of systematic perturbations on the system dynamics are also shown. To this aim, different PSA on the Schlögl and Ras/cAMP/PKA models were performed, in which one (PSA-1D), two (PSA-2D) or three (PSA-3D) parameters were simultaneously varied within given sweep intervals (chosen with respect to a fixed reference value for each parameter). Within these ranges, the numerical values of each varied parameter were determined with a linear sampling for the amounts of molecular species; a logarithmic sampling was instead considered for stochastic constants (if not stated otherwise), in order to uniformly span over many orders of magnitude. All PSA were performed by generating a set of different initial conditions — corresponding to different parameterizations of the model under investigation — and then automatically executing the parallel stochastic simulations with cuTauLeaping.

Computational results

To analyze the performances of cuTauLeaping, the same simulations executed on the GPU were carried out on a CPU architecture by exploiting the tau-leaping algorithm implemented in the software COPASI (version 4.8 Build 35, running on Windows 7 64-bit) [140]. COPASI has been recently integrated with a server-side tool, named Condor [159], that handles COPASI jobs, automatically splits them in sub-jobs and distributes the calculations on a cluster of heterogeneous machines; in this thesis this possibility is not used, as COPASI represents as a single-node CPU-bound reference implementation, which is currently single-threaded and does not exploit the physical and logical cores of the CPU.

The GPU used for the tests is a Nvidia GeForce GTX 590, a dual-GPU video card equipped with 2×16 streaming multiprocessors for a total of 1024 cores (cuTauLeaping automatically distributes the workload on the available SMXs); the performances were compared with a quad-core CPU Intel Core i7-2600 with a clock rate of 3.4 GHz.

In all simulations, the total simulation time $t_{\texttt{MAX}}$ for MM, PGN and Schlögl models was set to 5, 50 and 10 a.u., respectively, while for the Ras/cAMP/PKA model it was set to 150 a.u.; for each simulation, 100 samples were stored of all the molecular species occurring in the system. The value of the error control parameter of tau-leaping was set to $\epsilon = 0.03$, as suggested in [46].

The results of the comparison between cuTauLeaping and COPASI CPU tau-leaping are summarized in Table 5.2, which reports the running time (in sec) obtained by executing different batches of simulations of each model; these results were obtained using the RNG MRG32K3A in cuTauLeaping (see Section 4.2). Table 5.2 clearly shows the advantage of cuTauLeaping as the number of simulations increases. Interestingly, because of the architectural differences and the different clock rates, a single run of cuTauLeaping may be slower than the CPU counterpart, and it becomes fully profitable only by running multiple simulations. For instance, in the case of the Ras/cAMP/PKA model (Figure 5.23d), the break-even is reached around 2^7 simulations.

Thus, when less than ≈ 100 simulations of this specific pathway are needed, the use of a CPU implementation may be more convenient. Nonetheless, statistical analyses of

Table 5.2: Comparison of computational time of COPASI CPU tau-leaping and cuTauLeaping.

Model	*Simulations*	*CPU time*	*GPU time**	*Speedup*
Michaelis-Menten	2^6	0.047	0.038	1.23 ×
	2^8	0.219	0.046	4.75 ×
	2^{10}	0.905	0.047	19.25 ×
	2^{12}	4.087	0.051	80.14 ×
	2^{14}	19.001	0.073	260.29 ×
	2^{16}	76.691	0.172	445.87 ×
	2^{18}	309.241	0.530	583.47 ×
Prokaryotic gene network	2^6	0.468	0.120	3.90 ×
	2^8	1.997	0.121	16.50 ×
	2^{10}	8.112	0.124	65.42 ×
	2^{12}	32.807	0.128	256.30 ×
	2^{14}	130.885	0.175	747.91 ×
	2^{16}	526.535	0.591	890.92 ×
	2^{18}	2095.48	2.18	961.23 ×
Schlögl	2^6	0.202	0.723	0.28 ×
	2^8	0.811	0.875	0.92 ×
	2^{10}	3.603	0.979	3.68 ×
	2^{12}	13.993	1.156	12.10 ×
	2^{14}	56.254	1.578	35.64 ×
	2^{16}	224.454	3.534	63.51 ×
	2^{18}	905.664	10.163	89.11 ×
Ras/cAMP/PKA	2^6	118.873	320.1	0.37 ×
	2^8	445.632	322.4	1.38 ×
	2^{10}	1769.58	372.4	4.75 ×
	2^{12}	8828.05	551.4	16.01 ×
	2^{14}	35027.9	1530.1	22.89 ×
	2^{16}	133733	5482	24.39 ×
	2^{18}	534932**	21470	24.92 ×

*Exploiting the MRG32K3A random numbers generator

**Estimated value

stochastic temporal evolutions of biological systems require large batches of simulations (usually $\gg 100$) to derive statistically significant measures of the analyzed system dynamics. Note that, in general, the analytical determination of the break-even for an arbitrary model is a hard task, because it depends on its size (the number of reactions and molecular species) as well as on its parameterization, that might lead to stiffness phenomena, able to affect the running time of the used simulation algorithm. Moreover,

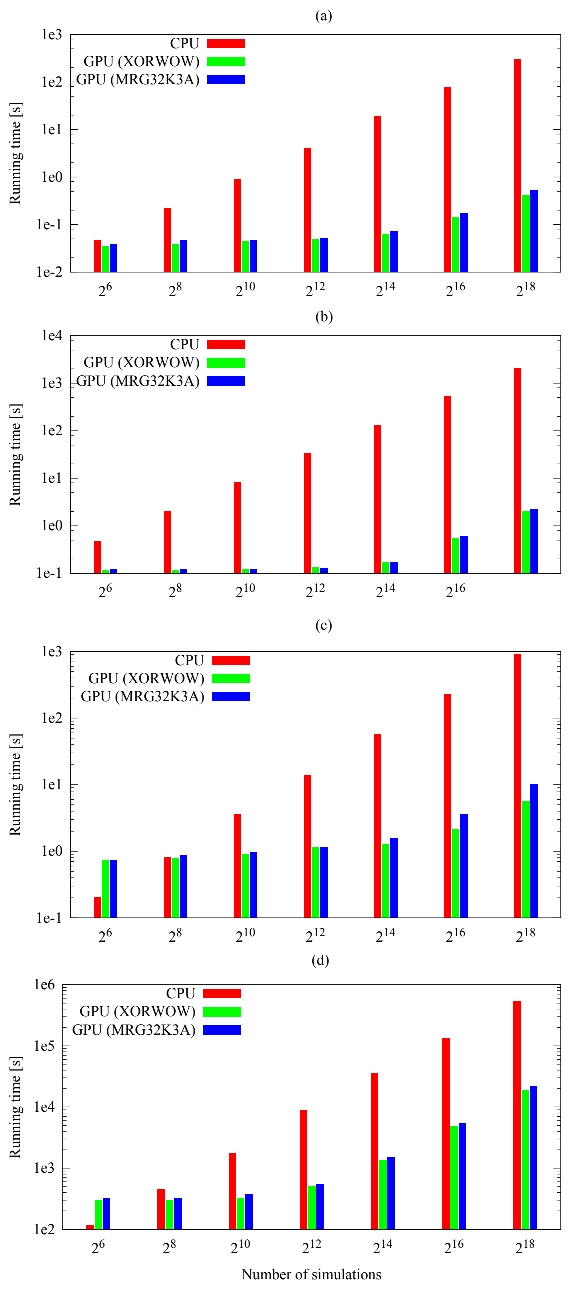


Figure 5.23: Comparison between the computational time taken by cuTauLeaping and COPASI CPU tau-leaping to execute different batches of stochastic simulations of the MM model (**a**), the PGN model (**b**), the Schlögl model (**c**), and the Ras/cAMP/PKA pathway (**d**) (see Appendix A). For each model, cuTauLeaping becomes more profitable than the CPU counterpart when a certain number of parallel simulations is run, with a break-even that depends on the complexity of the system. Considering the speedup, the best results achieved with cuTauLeaping — with respect to COPASI — are around 583× for the MM model, 961× for the PGN model, 90× for the Schlögl model, and 25× for the model of the Ras/cAMP/PKA pathway (see also Table 5.2).

if a biological system is characterized by multistability or by very large fluctuations in the dynamics of some molecular species (e.g., those occurring in low amounts), simulations running in different threads can be characterized by a high divergence of the execution flow, thus resulting in reduced parallelism and, consequently, in worse performances. For instance, if two threads simulating the same system reach very different system states, they can take different branches within the code (e.g., due to a rejection of an invalid putative state $\mathbf{x}'$ of the system), an event that can greatly affect the performance of the GPU.

For the sake of completeness, Figure 5.23 shows the running times of cuTauLeaping obtained by using the two RNGs implemented in cuTauLeaping, XORWOW and MRG32K3A, and compares these results with the running time of COPASI CPU tau-leaping. As described in Section 4.2, XORWOW presents statistical flaws and therefore should not be exploited to carry out Monte Carlo simulations. Thus, MRG32K3A is used as default RNG for cuTauLeaping. A more detailed comparison of these two RNGs — from both the points of view of quality and performances — is provided later in this section.

Finally, in order to investigate the influence of the size of the simulated system on cuTauLeaping performances, several tests on randomly generated synthetic models (RGSM) were executed. In particular, six distinct parameterizations of 100 different RGSM, each one consisting of 35 reactions and 33 species, were analyzed. The rationale behind the choice of this model size and various initial conditions was to compare the computational costs of cuTauLeaping for the simulation of RGSM on the one side, and the Ras/cAMP/PKA model on the other side. RGSM were generated according to the methodology proposed by Komarov *et al.* [165], and following two different strategies for the selection of the values of the stochastic constants. The results of these tests are given in Table 5.3 where, for each synthetic model, the average running times (given in sec) were evaluated by executing 2^{10} parallel simulations with $t_{\mathtt{MAX}} = 150$ a.u., to allow a direct comparison with the results listed in Table 5.2.

In tests 1 and 2, the values of stochastic constants of the RGSM were randomly selected with a uniform probability in $[0, 1]$; the initial molecular amounts were set to 10^3 (test 1) and 10^4 (test 2) for all species appearing in the systems. Both tests show that, on average, the computational time required for 2^{10} parallel simulations of RGSM is much lower than the time needed for the same number of parallel simulations of the Ras/cAMP/PKA model. In addition, it is clear that the initial molecular amounts considered in the parameterizations actually influence the results; in general, higher quantities lead to higher average running times.

Table 5.3: Running times of cuTauLeaping for the simulation of randomly generated synthetic models.

Test n.	*Model size*	*Initial molecular amounts*	*Stochastic constants interval*	*Average running time (standard deviation)*
1	35×33	$\mathbf{x} = (10^3, \ldots, 10^3)$	$[0, 1]$	1.906 (1.396)
2	35×33	$\mathbf{x} = (10^4, \ldots, 10^4)$	$[0, 1]$	8.149 (7.866)
3	35×33	$\mathbf{x} = (10^3, \ldots, 10^3)$	$[10^{-6}, 10]$*	1.436 (0.7)
4	35×33	$\mathbf{x} = (10^4, \ldots, 10^4)$	$[10^{-6}, 10]$*	8.263 (6.053)
5	35×33	$\mathbf{x} = (10^3, \ldots, 10^3)$	$[10^{-9}, 10^4]$*	1.247 (0.543)
6	35×33	$\mathbf{x} = (10^4, \ldots, 10^4)$	$[10^{-9}, 10^4]$*	18.509 (91.976)
7	80×80	$\mathbf{x} = (10^3, \ldots, 10^3)$	$[10^{-6}, 10]$*	13.242 (10.623)
8	80×80	$\mathbf{x} = (10^4, \ldots, 10^4)$	$[10^{-9}, 10^4]$*	96.133 (302.05)

*Logarithmic sampling

In tests 3 to 6 a modified strategy was exploited to select the values of stochastic constants, which were logarithmically sampled in the given range in order to uniformly span over different orders of magnitude. To obtain more realistic parameterizations, in tests 3 and 4 this range was defined according to the smallest and to the highest values of stochastic constants appearing in the Ras/cAMP/PKA model (see Table A.8 in Appendix A); in tests 5 and 6, the range was widened to six orders of magnitude larger than the values used in tests 3 and 4. Also in these cases, the average running time required for the simulation of the RGSM is lower than the time needed to execute the same number of parallel simulations of the Ras/cAMP/PKA model. Moreover, the running time is mainly influenced by the initial molecular amounts rather than the values of the stochastic constants used in the different parameterizations.

In addition, in test 7 and 8, 2^{10} parallel stochastic simulations of RGSM, whose sizes are approximately 4× bigger than the Ras/cAMP/PKA model, were performed. Even considering the worst result (test 8), the average running time required to simulate these larger synthetic models is still low, suggesting that the system size has not a direct impact on the performances of cuTauLeaping, while the complexity of the system — such as the presence of positive or negative feedbacks possibly leading to oscillatory dynamics (as in the Ras/cAMP/PKA model), or reactions that lead to stiffness (as in the Schlögl model) — is much more relevant.

Comparison of quality and performances using RNGs XORWOW and MRG32K3A

The XORWOW algorithm [200] is known to present statistical flaws [133] and therefore should not be exploited to perform Monte Carlo simulations. Nevertheless, it is more performant than the other RNGs available in CURAND and, as such, it might be used when a strong reduction of the computational costs is fundamental.

The aim of the tests presented in this section was to compare the outcomes of stochastic simulations when using XORWOW instead of MRG32K3A [177]. This comparison was carried out on the Schlögl model by executing $2^{10}, 2^{12}, 2^{14}, 2^{16}$ parallel simulations and calculating the frequency distributions of the chemical species X at time $t = 10$ a.u.. The results of these four tests are shown in Figure 5.24.

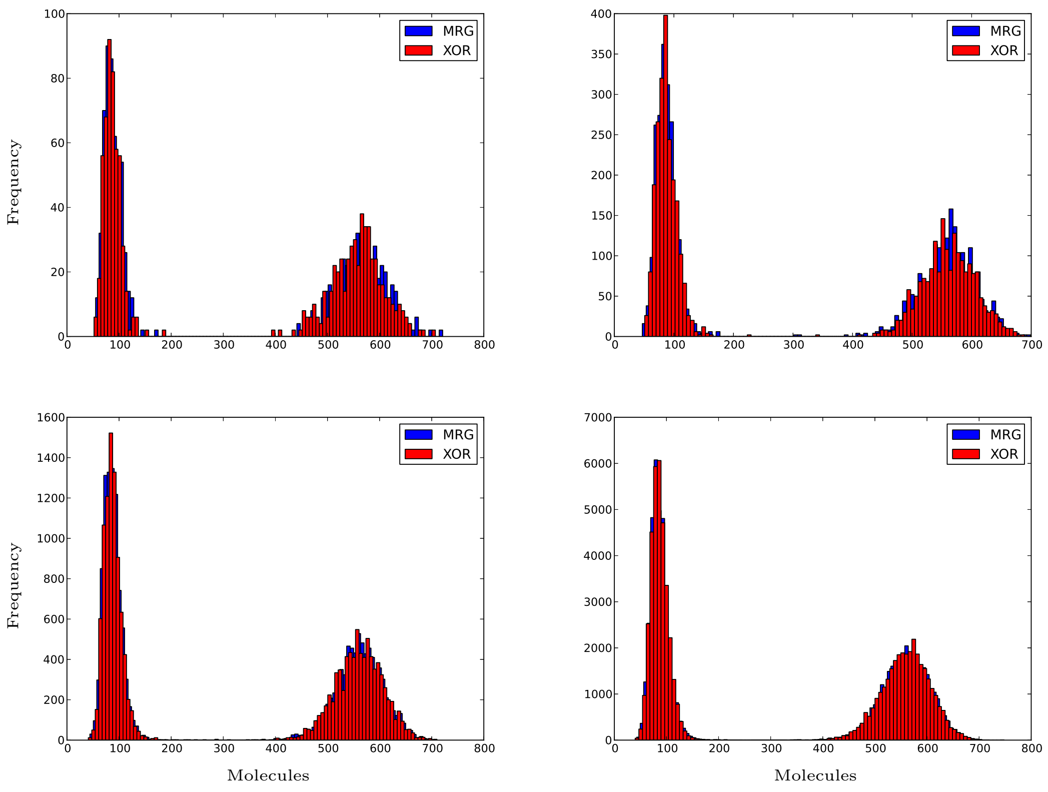


Figure 5.24: Comparison of the frequency distribution of the chemical species X of the Schlögl model at time $t = 10$, using the MRG32K3A (blue) and the XORWOW (red) RNGs. Top-left: 2^{10} simulations; top-right: 2^{12} simulations; bottom-left: 2^{14} simulations; bottom-right: 2^{16} simulations.

In order to verify whether the samples corresponding to XORWOW and MRG32K3A were drawn from the same distribution, the Kolmogorov-Smirnov (K-S) [96] statistics was exploited. When the K-S statistic is small or the p-value is high, the hypothesis that the distributions of the two samples are the same cannot be rejected. Results in Table 5.4 confirm that, despite XORWOW presents statistical flaws, according to the K-S test the frequency distributions of X obtained with XORWOW and MRG32K3A can be considered completely equivalent. This is also shown in Figure 5.24, where it can observed that the distributions are perfectly overlapped.

Table 5.4: Kolmogorov-Smirnov statistics of the frequency distribution of species X in the Schlögl model.

Number of parallel simulations	*K-S test*	*p-value*
2^{10}	0.05	0.9994
2^{12}	0.04	0.9999
2^{14}	0.04	0.9999
2^{16}	0.05	0.9994

Table 5.5 reports the comparison of the overall running time (expressed in sec) of different batches of simulations for the four models described in Appendix A, executed using either XORWOW or MRG32K3A. These results show that XORWOW outperforms MRG32K3A in all cases.

Altogether, these results show that, despite XORWOW presents statistical flaws and therefore should not be safely exploited to carry out Monte Carlo simulations, the frequency distributions of X obtained with XORWOW and MRG32K3A can be considered equivalent; in addition, XORWOW allows to achieve a higher speedup with respect to MRG32K3A. Nevertheless, in order to have high quality simulated dynamics and reasonable performances, MRG32K3A is used as default RNG for this thesis.

Analysis of bistability in the Schlögl model

Bistability is a capacity exhibited by many biological systems, consisting in the possibility of switching between two different stable steady states in response to some chemical signaling (see, e.g., [64, 263, 353] and references therein). The Schlögl model is one of the simplest prototypes of chemical systems presenting a bistable dynamic behavior [342, 354]. This system is characterized by the fact that, starting from the same initial conditions, its dynamics can reach either the low or the high steady state; switches between the two steady states can also occur due to stochastic fluctuations.

cuTauLeaping can be efficiently exploited for the execution of a massive number of simulations of the Schlögl model, with the same initial parameterization, in order to produce a frequency distribution of the molecular species that exhibits the bistable

Table 5.5: Computational performance of cuTauLeaping using different random numbers generators.

Model	*Simulations*	*CPU time*	*GPU time* (*XORWOW*)	*GPU time* (*MRG32K3A*)	*Speedup* (*XORWOW*)	*Speedup* (*MRG32K3A*)
Michaelis-Menten model	2^6	0.047	0.035	0.038	1.34 ×	1.23 ×
	2^8	0.219	0.038	0.046	5.75 ×	4.75 ×
	2^{10}	0.905	0.044	0.047	20.56 ×	19.25 ×
	2^{12}	4.087	0.049	0.051	83.41 ×	80.14 ×
	2^{14}	19.001	0.064	0.073	296.89 ×	260.29 ×
	2^{16}	76.691	0.141	0.172	543.91 ×	445.87 ×
	2^{18}	309.241	0.406	0.530	761.68 ×	583.47 ×
Prokaryotic gene network	2^6	0.468	0.117	0.120	4.00 ×	3.90 ×
	2^8	1.997	0.117	0.121	17.07 ×	16.50 ×
	2^{10}	8.112	0.123	0.124	65.95 ×	65.42 ×
	2^{12}	32.807	0.132	0.128	248.54 ×	256.30 ×
	2^{14}	130.885	0.172	0.175	760.96 ×	747.91 ×
	2^{16}	526.535	0.55	0.591	957.34 ×	890.92 ×
	2^{18}	2095.48	2.05	2.18	1022.2 ×	961.23 ×
Schlögl model	2^6	0.202	0.725	0.723	0.27 ×	0.28 ×
	2^8	0.811	0.790	0.875	1.02 ×	0.92 ×
	2^{10}	3.603	0.896	0.979	4.02 ×	3.68 ×
	2^{12}	13.993	1.132	1.156	12.36 ×	12.10 ×
	2^{14}	56.254	1.265	1.578	44.46 ×	35.64 ×
	2^{16}	224.454	2.113	3.534	106.22 ×	63.51 ×
	2^{18}	905.664	5.644	10.163	160.46 ×	89.11 ×
Ras/cAMP/PKA pathway	2^6	118.873	303.5	320.1	0.39 ×	0.37 ×
	2^8	445.632	299.56	322.4	1.49 ×	1.38 ×
	2^{10}	1769.58	327.4	372.4	5.40 ×	4.75 ×
	2^{12}	8828.05	515.7	551.4	17.12 ×	16.01 ×
	2^{14}	35027.9	1376.5	1530.1	25.45 ×	22.89 ×
	2^{16}	133733	4898	5482	27.30 ×	24.39 ×
	2^{18}	534932*	19004	21470	28.15 ×	24.92 ×

*Estimated value

behavior. Generally speaking, this kind of investigation allows the implicit identification of attractors and multiple steady states of a system, and helps to (empirically) determine the probability of reaching a particular state during the dynamical evolution of the system itself.

To this aim, a total of 2^{18} simulations of the Schlögl model were performed, keeping track of the molecular amount of species X in each simulation, sampled in 100 time instants uniformly distributed over the simulation time. In particular, to detect the initial jump either to the low or to the high steady states, that takes place in the first time instants of the simulations, 2^{17} simulations with $t_{\texttt{MAX}} = 3$ a.u. were initially performed; then, for a deeper investigation of the bistable switching behavior of the system, the other simulations were executed considering $t_{\texttt{MAX}} = 150$ a.u.. The results of simulations were used to calculate the histograms of the molecular amount of X, that were then exploited to realize a heatmap showing the frequency distribution of this species between the two stable steady states. Figure 5.25a shows the initial transient of the dynamics, where a slightly higher probability to reach the low steady state can be observed, starting from the initial configuration of the Schlögl system (described in Appendix A, Tables A.4 and A.5). Figure 5.25b shows the frequency distribution of reaching either the low or the high steady state (around 100 and 600 molecules of species X, respectively), highlighting a larger variance concerning the high steady state.

In Figure 5.26 it is shown the frequency distribution of molecular amounts of X in perturbed conditions of the Schlögl model, evaluated by exploiting a PSA-1D in which the value of the stochastic constant c_3 is uniformly varied in the interval $[6.9 \cdot 10^{-4}, 1.4 \cdot 10^{-3}]$. The frequency distribution was calculated according to 2^{18} simulations, where the dynamics was sampled at the single time instant $t = 10$ a.u., according to ten different values of the stochastic constant c_3 in the chosen sweep range. The total running time to execute this PSA-1D was 34.92 sec with cuTauLeaping, and 1013.51 sec with COPASI, thus achieving a 116× speedup. Figure 5.26 shows that increasing values of c_3 induce a decrease (increment, respectively) in the frequency distribution of X concerning the low (high, respectively) steady state, whereas for intermediate values of c_3 the system is characterized by an effective bistable behavior.

Finally, Figure 5.27 shows the results of a PSA-3D performed by simultaneously varying the values of the stochastic constants c_1, c_2 and c_3 in the ranges $[2.9 \cdot 10^{-7}, 3.1 \cdot 10^{-7}]$, $[8.0 \cdot 10^{-5}, 1.1 \cdot 10^{-4}]$ and $[6.0 \cdot 10^{-4}, 1.2 \cdot 10^{-3}]$, respectively. For each stochastic constant, taken independently from the others, the chosen range corresponds to a condition of effective bistability of the Schlögl model. The values of the three stochastic

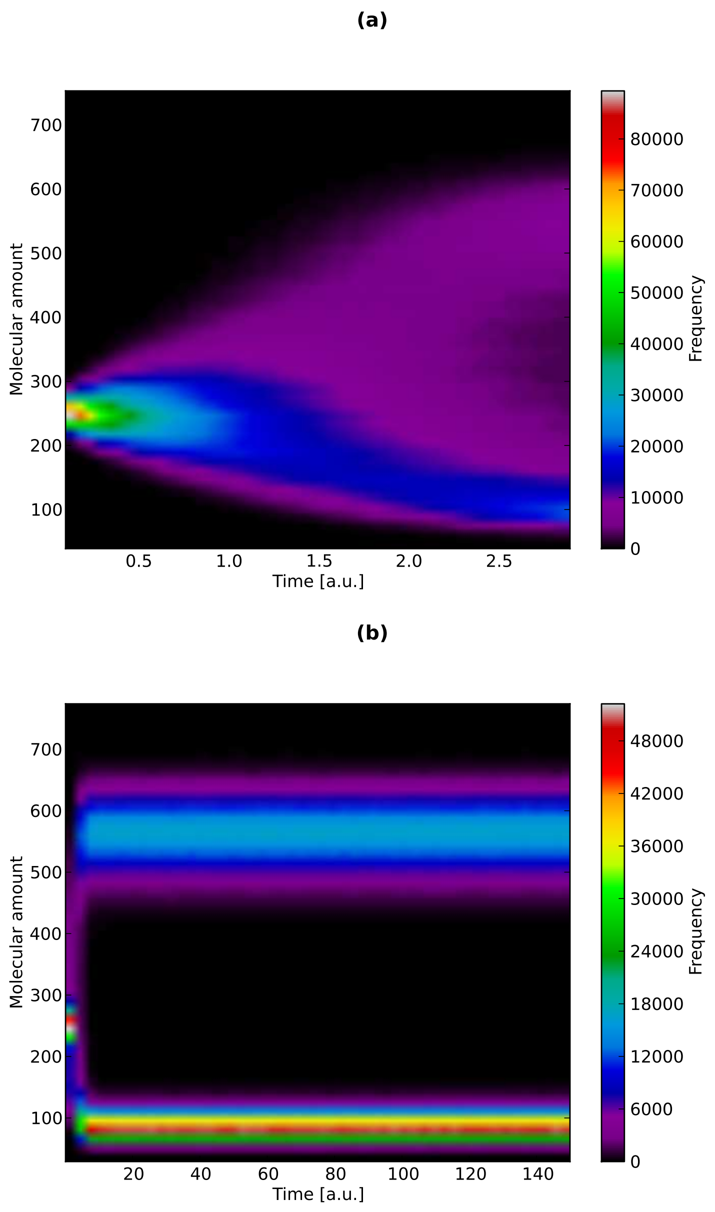


Figure 5.25: Frequency distribution of the molecular amount of molecular species X in the Schlögl model, calculated using a total of 2^{18} parallel simulations executed by cuTauLeaping. **(a)** Plot of the frequency distribution of X considering $t_{\texttt{MAX}} = 3$ a.u., to detect the bistable switching behavior that takes place in the first time instants of the dynamics; a slightly higher probability to reach the low steady state can be observed, starting from the initial state of the Schlögl system. **(b)** Plot of the frequency distribution of X considering $t_{\texttt{MAX}} = 150$ a.u., to investigate the stability of the two steady states of the system; the heatmap highlights the two stable states (around 100 and 600 molecules of species X), and shows larger stochastic fluctuations around the high steady state.

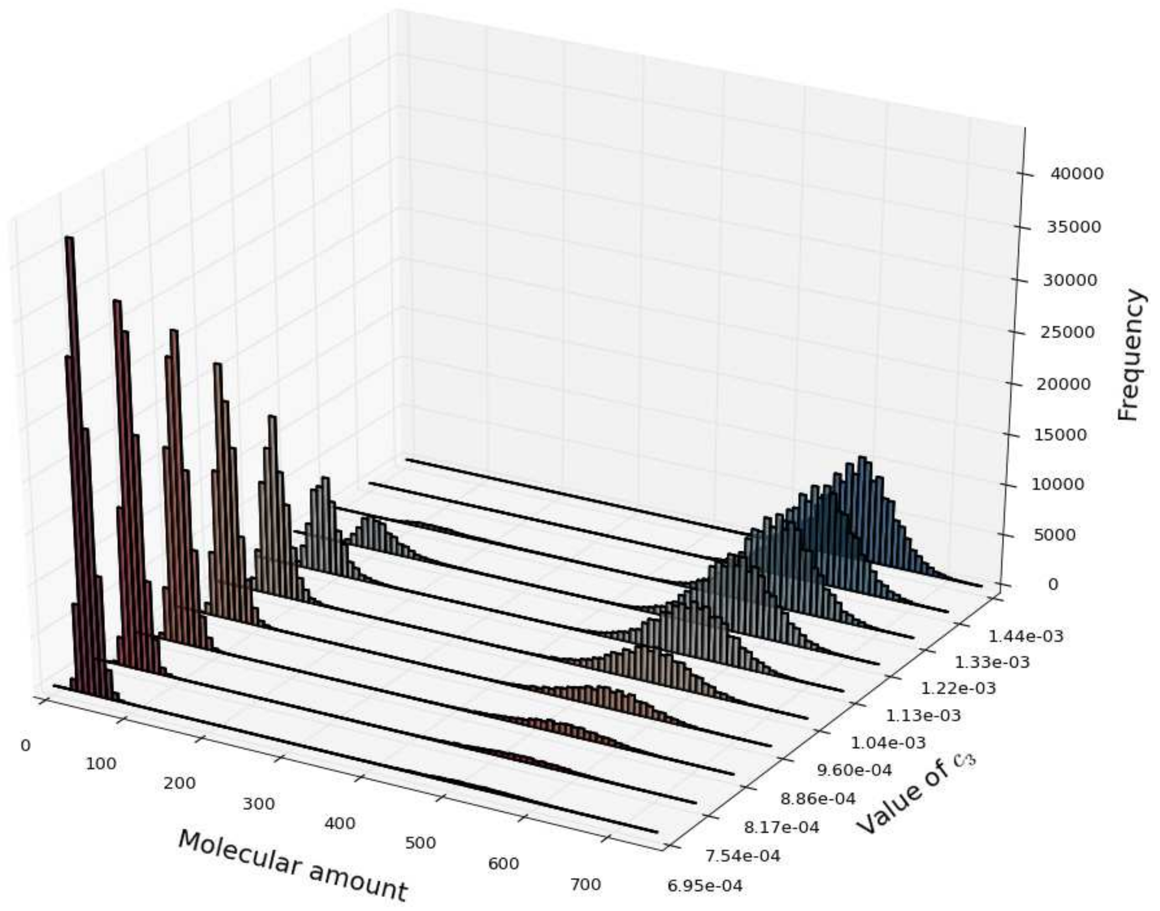


Figure 5.26: Results of a PSA-1D on the Schlögl model, in which the value of the stochastic constant c_3 is varied in the interval $[6.9 \cdot 10^{-4}, 1.4 \cdot 10^{-3}]$ (the set of reactions and the values of all other parameters are given in Appendix A, Tables A.4 and A.5). Each frequency distribution is calculated according to 2^{18} simulations executed by cuTauLeaping, measuring the amount of the molecular species X at the time instant $t = 10$ a.u., considering ten different values of the stochastic constant c_3 within the sweep interval. The figure shows that increasing values of c_3 induce a decrease (increment) in the frequency distribution of X concerning the low (high) steady state, with intermediate values of c_3 characterized by an effective bistable behavior.

constants were uniformly sampled in a $16 \times 16 \times 16$ three-dimensional lattice; for each sample, 256 simulations are executed (for a total of 2^{20} simulations) and the frequency distribution of the amount of species X at the time instant $t = 10$ a.u. is evaluated. This set of values was then partitioned according to the reached (low or high) stable steady state; in Figure 5.27, the red (blue, respectively) region corresponds to the parameterizations of the model which yield the high (low, respectively) steady state most frequently. The green region represents a set of conditions whereby both steady states are equally reached.

PSA of the Ras/cAMP/PKA model

In this section the results of a PSA carried out on the stochastic model of the Ras/-cAMP/PKA pathway [27, 255] are reported. In [27], it was shown that intrinsic noise within the Ras/cAMP/PKA pathway can enhance the robustness of the system in response to different perturbations of the model parameters, ensuring the presence of stable oscillatory regimes, as previously investigated for other biological systems (see [321] and references therein). Indeed, stochastic simulations of the functioning of this pathway showed that yeast cells might be able to respond appropriately to an alteration of some basic components — such as the intracellular amount of pivotal proteins, that can be related to the stress level [210, 348] — fostering the maintenance of stable oscillations during the signal propagation. This behavior might suggest a stronger adaptation capability of yeast cells to various environmental stimuli or endogenous variations.

In [27, 255], in particular, it was shown that the intracellular pool of guanine nucleotides (GTP, GDP), as well as the molecular amounts of protein Cdc25 — that positively regulates the activation of Ras protein, and that is negatively regulated by PKA — are both able to govern the establishment of oscillatory regimes in the dynamics of the second messengers cAMP and of protein PKA. In turn, this behavior can influence the dynamics of downstream targets of PKA, such as the periodic nucleocytoplasmic shuttling of the transcription factor Msn2 [95, 210]. In addition, in [27, 255] it was highlighted that stochastic and deterministic simulations of the Ras/cAMP/PKA pathway can yield qualitatively different outcomes: in some conditions, characterized by very low amounts of pivotal proteins in this pathway (e.g., Cdc25), the stochastic approach provides stable oscillatory regimes of cAMP, while the deterministic approach shows damped oscillations. Therefore, these results remark the role played by noise in the Ras/cAMP/PKA pathway and the usefulness of executing stochastic simulations.

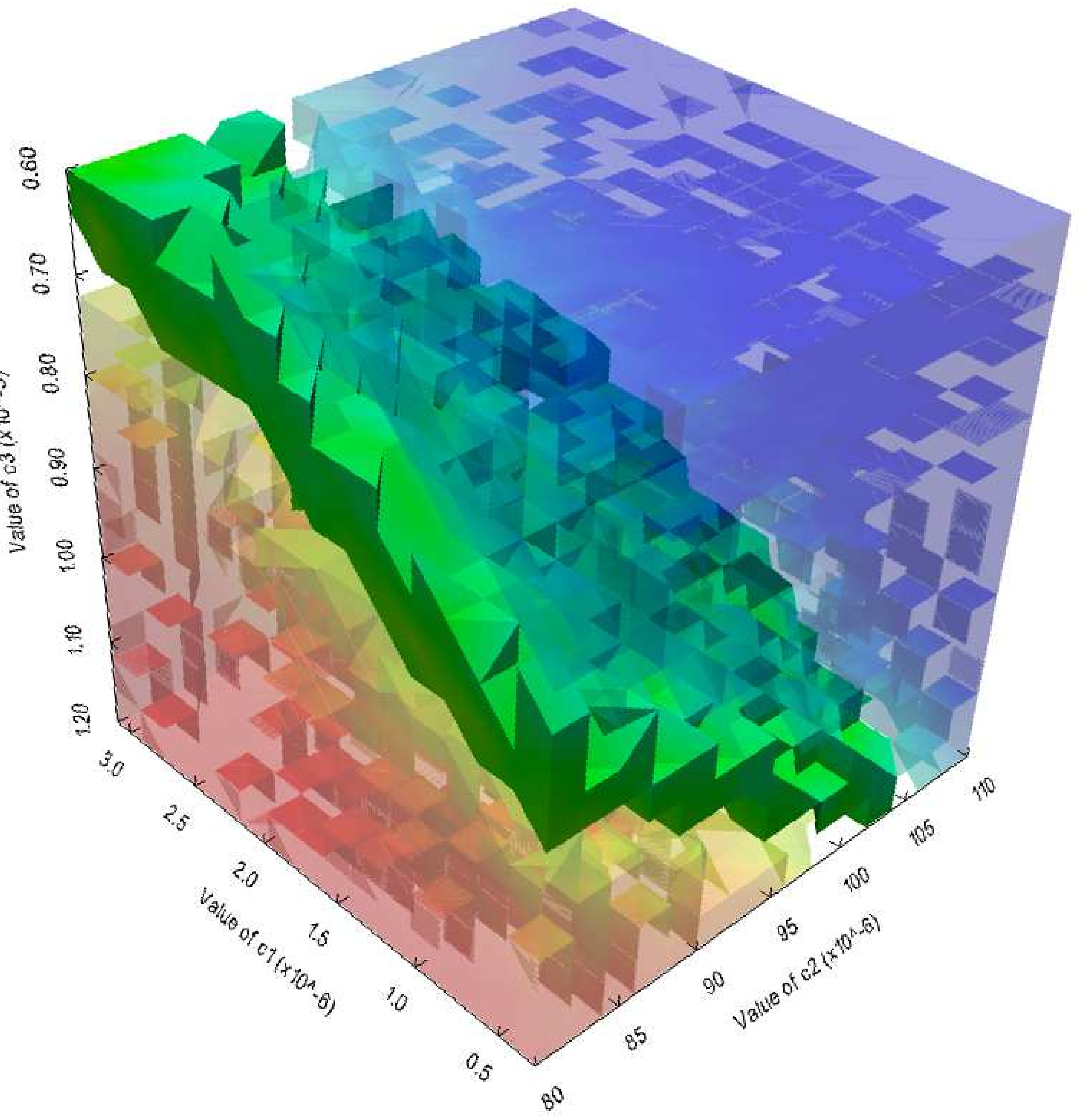


Figure 5.27: Results of a PSA-3D on the Schlögl model, performed by varying the stochastic constants c_1, c_2 and c_3 in the intervals $[2.9 \cdot 10^{-7}, 3.1 \cdot 10^{-7}]$, $[8.0 \cdot 10^{-5}, 1.1 \cdot 10^{-4}]$ and $[6.0 \cdot 10^{-4}, 1.2 \cdot 10^{-3}]$, respectively. The values of the stochastic constants were uniformly sampled in a $16 \times 16 \times 16$ three-dimensional lattice; for each sample, 256 simulations were executed with cuTauLeaping (for a total of 2^{20} simulations) and evaluated the frequency distribution of the amount of the molecular species X at the time instant $t = 10$ a.u. was evaluated. This set of values was then partitioned according to the reached (low or high) stable steady state; in the plot, the red (blue) region corresponds to the parameterizations of the model which yield the high (low) steady state most frequently. The green region represents a set of conditions whereby both steady states are equally reached.

To deeply investigate the role played by guanine nucleotides and Cdc25, in this work it is further analyzed the extended version of the Ras/cAMP/PKA model presented in [27, 255], where the reactions responsible for the occurrence of oscillatory behaviors were included. To this aim, a PSA-2D was performed to simulate the system dynamics in perturbed conditions, where it was simultaneously varied the amount of GTP in the interval $[1.9 \cdot 10^4, 5.0 \cdot 10^6]$ molecules (corresponding to a reduced nutrient availability, up to a normal growth condition) and the amount of Cdc25 in the interval $[0, 600]$ molecules (ranging from the deletion to a 2-fold overexpression of these regulatory proteins). A total of 2^{16} different initial parameterizations were uniformly distributed over this bidimensional parameter space. In Figure 5.28a the amplitude of cAMP oscillations is shown in each of these initial conditions, where an amplitude value equal to zero corresponds to a non oscillating dynamics; the amplitude values of cAMP oscillations were calculated as described in [255].

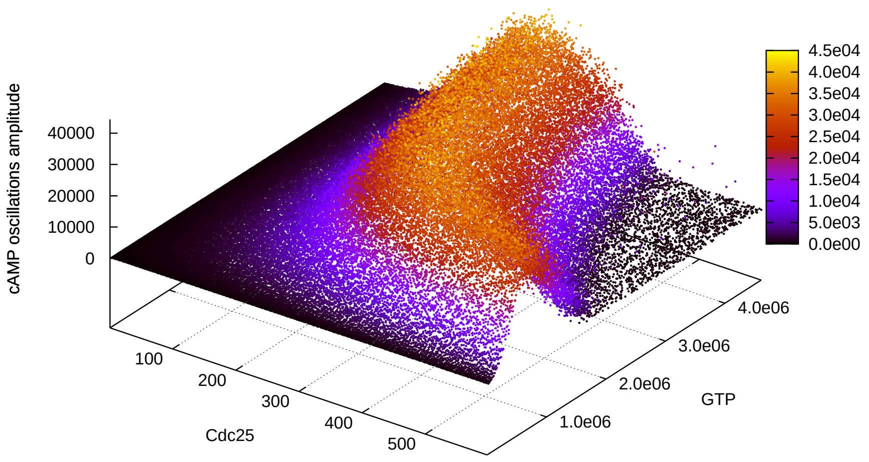
This figure shows that oscillatory regimes are established for basically any value of GTP when the amount of Cdc25 is at normal condition or slightly lower, while if the amount of Cdc25 increases, no oscillations of cAMP occur when GTP is high, but oscillatory regimes are still present if GTP is low. In order to compare the advantage of using cuTauLeaping to perform this stochastic analysis with respect to a CPU implementation, in Figure 5.28b it is shown a previous analysis performed on the CPU [27], which was obtained with a comparable computational time, albeit in the case of the CPU-based analysis only 2^8 parameterizations of the Ras/cAMP/PKA pathway (corresponding to 2^8 independent simulations) could be analyzed.

These computational results can suggest possible interesting behaviors of the biological system under investigation. In this case, for instance, the establishment of oscillatory regimes in the above mentioned conditions can be due to the fact that Ras proteins are more frequently in their inactive state (that is, loaded with GDP instead of GTP) when the ratio GTP/GDP decreases. Since in normal growth conditions the concentration of GTP is 3 to 5 times higher than GDP, the decreased activity of Ras proteins in the considered perturbed conditions — which are characterized by a favored unproductive binding/unbinding with GDP — can induce the establishment of an oscillatory regime (see also [27] for more details).

5.3.3 Discussion

To reduce the computational costs related to the analyses of mathematical models of real biological systems, two conceptually simple ways can be considered to parallelize stochastic simulations. The easiest solution consists in generating multiple threads on

(a)



(b)

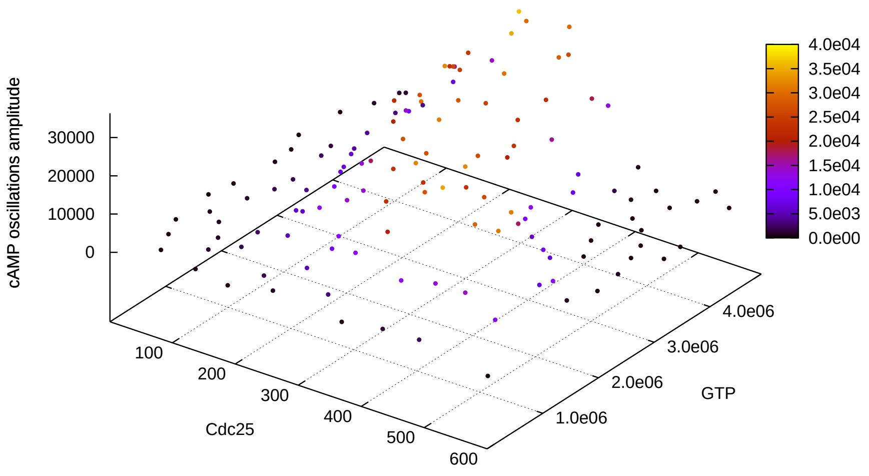


Figure 5.28: Results of a PSA-2D on the Ras/cAMP/PKA model by varying the amount of GTP in the interval $[1.9 \cdot 10^4, 5.0 \cdot 10^6]$ molecules (ranging from a reduced nutrient availability to a normal growth condition), and the amount of Cdc25 in the interval $[0, 600]$ molecules (ranging from the deletion to a 2-fold overexpression of this GEF proteins). The figure shows the amplitude of cAMP oscillations, evaluated as described in [255]; an amplitude value equal to zero corresponds to a non oscillating dynamics. (**a**) Plot of the results obtained by running 2^{16} parallel simulations with cuTauLeaping; (**b**) plot of the results obtained by running 2^8 sequential simulations, performed on the CPU. The two batches of parallel and sequential simulations were executed with a comparable computational time.

multi-core workstations, but it immediately turns out to be undersized, since the number of cores on high-end machines can be far lower than the number of simulations required for computational analysis as PE, PSA, sensitivity analysis, RE and ED. The other way consists in distributing the stochastic simulations on a cluster of machines, which may as well result inadequate for several problems. First of all, it is economically expensive and very power-demanding; secondly, it takes a dedicated software infrastructure to handle workload balancing, network communication and the possible errors due to nodes downtime or server-node communication issues; thirdly, if the nodes of the cluster are heterogeneous, the slowest machines may represent a bottleneck for the whole task. In addition, a cluster implementation may not always scale well because of two problems: on the one hand, the speedup is approximately proportional to the number of independent simulations that run on a dedicated node (i.e., a million nodes for common tasks like PE and SA); on the other hand, the running time of each simulation can be larger than the overhead requested for server-node communication.

An alternative methodology to perform multiple and massively parallel simulations consists in exploiting the GPGPU architecture. The modern GPU of mid-range price contains thousands of cores that — as long as the computational task can be subdivided and optimized for a SIMD architecture — allow an impressive peak of computational power, and also a higher energetic efficiency with respect to an equivalent CPU-based solution.

cuTauLeaping was developed taking into account all these aspects, and it represents an implementation of tau-leaping algorithm as a set of strongly optimized CUDA kernels, able to simultaneously execute multiple independent simulations on a single machine. The specific design of cuTauLeaping presents some additional advantages: it avoids any memory transfer to and from the host, thus reducing the overall running time, and it can be embedded into the GPU-based software framework described in this thesis, that was developed for PE [231, 232], RE [234] and ED [236]. As a matter of fact, the modularity of the implementation and the mutual independence of the multiple simulations allows to easily wrap cuTauLeaping with any other methodology that needs or can benefit from these massive parallel executions.

To achieve even better performances, the tau-leaping algorithm was redesigned to (*i*) avoid the conditional branches, thus exploiting the underlying SIMD architecture as much as possible, and (*ii*) capitalize on the CUDA's memory hierarchy, by pre-calculating some of the needed data structures and by allocating the most used ones into the fastest, yet smallest, memories. Indeed, one the biggest limiting factor for a good occupancy of the CUDA resources is a large use of the shared memory, which can improve the overall performances at the cost of reducing the theoretical occupancy of the SMX. Table 5.6 lists the data type and size of the vectors used by cuTauLeaping. For performances reasons, these vectors are stored into the high-performance memories: vectors containing information that change during the simulation are allocated into the shared memory, while the other information are stored into the constant memory. The dimensions of these vectors are proportional to the number of threads forming a block (T), the number of reactions (M), the number of molecular species present in the system (N) and the number of non-zero entries (Z) of the corresponding non-flattened stoichiometric matrix.

Table 5.6: tau-leaping data structures residing in CUDA high-performance memories.

Array name	*Data type*	*Array size*	*Memory type*
$\mathbf{x}$	unsigned int (4 bytes)	$N \times T$	shared memory
$\mathbf{x}'$	int (4 bytes)	$N \times T$	shared memory
$\mathbf{a}$	float (4 bytes)	$M \times T$	shared memory
$\mathbf{c}$	float (4 bytes)	$M \times T$	shared memory
$\mathbf{K}$	unsigned int (4 bytes)	$M \times T$	shared memory
$\boldsymbol{\chi}$	char (1 byte)	$M \times T$	shared memory
$\mathbf{A}$	uchar4 (4 bytes)	$O(Z)$	constant memory
$\mathbf{V}$	uchar4 (4 bytes)	$O(Z)$	constant memory
$\mathbf{V}^{\mathrm{T}}$	uchar4 (4 bytes)	$O(Z)$	constant memory
$\bar{\mathbf{V}}$	uchar4 (4 bytes)	$O(Z)$	constant memory

Specifically, according to the memory structures described in Table 5.6, in cuTauLeaping the exact shared memory consumption per SMX for a model consisting of N molecular species and M chemical reactions, simulated by T threads per block, is equal to $T \times [13(bytes) \times M(reactions) + 8(bytes) \times N(species)]$. Since the shared memory is a limited resource on the GPU, it follows that the maximum size of a block is proportional to the size of the system, leading to the upper bound

$$T \leq \left\lfloor \frac{MAX_{shared}}{13M + 8N} \right\rfloor, \tag{5.3}$$

where MAX_{shared} corresponds to the amount of shared memory available on each SMX for the specific architecture. According to Equation 5.3, on a GPU based on the Fermi architecture, the maximum size of a block for the MM, PGN, Schlögl and Ras/cAMP/PKA models corresponds to 692, 381, 646 and 63 threads, respectively. More generally, considering a theoretical, very large stochastic model composed by 100 reactions and 100 species, the maximum value for T would be 23; multiple blocks can be launched and run on different SMXs, if these are available on the GPU, thus allowing a further level of parallelism. For instance, the GeForce GTX 590 used in the tests is equipped with 32 SMXs and could therefore execute $23 \times 32 = 736$ simultaneous threads for this theoretical model. Since the shared memory represents a limiting factor for the parallelism, a subset of the data structures listed in Table 5.6 might be moved from the shared memory to the slower global memory, thus increasing the T value at the cost of higher latencies in the access to data and of higher computational costs. Being this a relevant aspect of the implementation, the optimization of these data structures is in progress, to the purpose of increasing the level of parallelism and further reducing the computational time.

In order to analyze the boost of performances, cuTauLeaping was compared with a standard CPU implementation (using the software COPASI [140] as reference), by running several identical simulations of four biological models of increasing size and complexity. Results showed that tau-leaping running on GPU yields much better results and becomes particularly profitable when a large number of simulations have to be performed. Interestingly, when the number of simulations is limited (ranging from a few units to around one hundred, for the four test models), the CPU version may result more efficient than the GPU, being the break-even between the two implementations directly dependent on the complexity of the system and on its emergent dynamics. It is worth noting that, although the computational speedup achieved with cuTauLeaping might be improved by exploiting a faster RNG, such as XORWOW, the results obtained by using the more reliable RNG MRG32K3A still show a relevant reduction of running times with respect to COPASI CPU tau-leaping (see Table 5.5). Therefore, cuTauLeaping represents an advantageous tool to carry out thorough computational analyses of stochastic biological systems that usually require a huge number of simulations.

In addition, different tests were performed on RGSM, suggesting that the inherent complexity of the system and the chosen parameterization are more important than the model size, and they can greatly affect the performances of the simulation algorithm. The variation of the initial parameterization, which is indispensable to carry out a perturbation analysis, usually induces quantitatively and qualitatively distinct dynamical behaviors. Even more important is the fact that different parameterizations generally result in different running times, leading to potentially huge and surely unfeasible computational costs, especially when standard CPU executions of stochastic simulation algorithms are performed. As an example to explain this important matter, the running time of cuTauLeaping and of COPASI CPU tau-leaping were compared when varying a single parameter in the Ras/cAMP/PKA model over 5 orders of magnitude (namely,

2^{10} simulations were executed varying the value of the stochastic constant c_3 in the sweep interval $[1.5 \cdot 10^{-3}, 1.5 \cdot 10]$). Figure 5.29 shows that, in this situation, the computational cost of tau-leaping running on CPU rapidly increases; this behavior could become prohibitive if several independent simulations need to be executed.

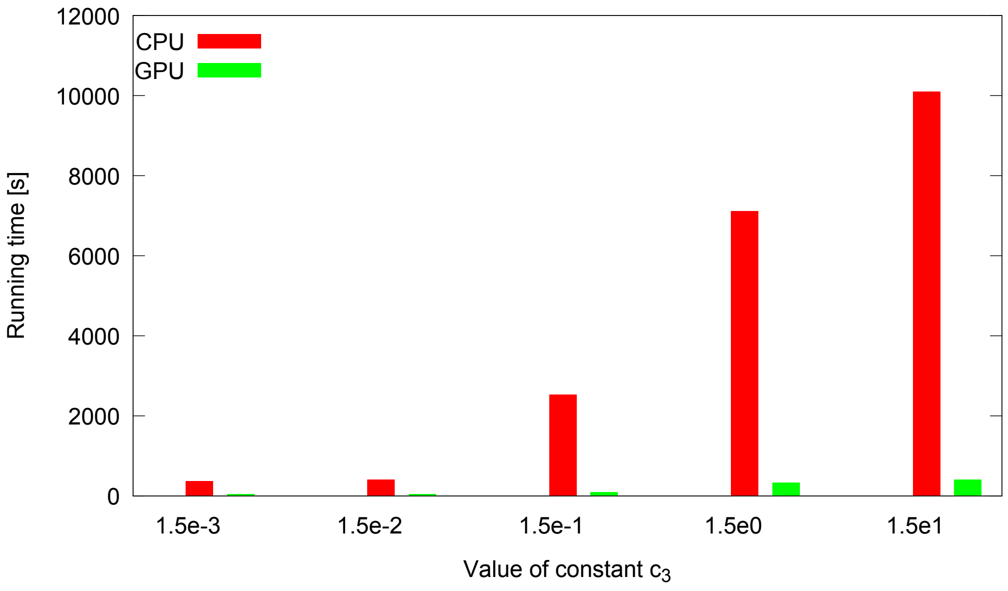


Figure 5.29: Running times of cuTauLeaping and COPASI CPU tau-leaping to execute a PSA-1D of the Ras/cAMP/PKA model, where the stochastic constant c_3 was varied in the interval $[1.5 \cdot 10^{-3}, 1.5 \cdot 10]$ and a total of 2^{10} simulations were executed. The plot shows how the computational cost of tau-leaping running on CPU rapidly increases; this behavior can become prohibitive if several independent simulations need to be executed. On the contrary, cuTauLeaping shows a very moderate increase in the running times and outperforms the CPU implementation of tau-leaping.

On the contrary, cuTauLeaping shows a very moderate increase in the running times, although following the same growth trend of the CPU counterpart, and outperforms the CPU implementation. An explanation for this behavior is that, being the CPU sequential, a simulation can start as long as the previous one terminated, whilst in the case of GPU the overall running time roughly corresponds to the running time of the slowest simulation. This is particularly relevant in the case of parameterizations leading to high running times on the CPU (e.g., when $c_3 = 1.5 \cdot 10$ in Figure 5.29), where the speedup granted by the use of cuTauLeaping is 25×, compared to the 4.75× speedup (see Table 5.2) achieved with the reference parameterization of the Ras/cAMP/PKA model (Appendix A, Table A.8).

A *fine-grain* GPU parallelization of tau-leaping was previously proposed in [165], to the aim of accelerating the execution of single runs of tau-leaping. In that work, the computational performances were discussed in relation to very large systems of molecular interactions (with better gains achieved for 10^5 reaction channels), and tested over a *synthetically* generated network consisting of $M = N = 1000$ reactions and species. Taking into account the above mentioned issues related to the model parameterization, there is a remarkable aspect that should not be left out when assessing the effective performances of fine-grain implementations of this type. Namely, the synthetic network used in [165] was characterized by a homogeneous initial parameterization (i.e., the values of all stochastic constants were randomly selected with a uniform distribution in $[0, 1]$), therefore largely limiting the biochemical meaning of the distinct reaction rates that very different molecular interactions — transcription rates, post-translational modification rates, diffusion rates, catalyzed processes, etc. — do actually present in real cellular systems [189, 249]. An arbitrary modification of these values, that are of pivotal importance in the definition and in the analysis of validated models of *real* biological systems, might possibly result in different computational performances of such fine-grain GPU accelerations whenever tested on other well assessed mathematical models of biological systems [180]. This is corroborated by the results that were obtained from the analyses of RGSMs, which altogether highlight the impact of the model parameterization on the computational performances: in particular, in cuTauLeaping the highest running times with large standard deviation values were obtained in tests 6 and 8 presented in Table 5.3, which are characterized by the largest intervals for the choice of stochastic constants and by the highest initial molecular amounts.

As a matter of fact, stochastic modeling and simulation methods are usually assumed to be suitable for relatively small systems (such as signaling pathways), consisting of a few tens of reactions and species, defined according to a bottom-up modeling approach whereby a mechanistic description of the most relevant molecular interactions is provided [320, 355]. The rationale behind this is that a good initial parameterization for models of this type cannot be usually settled by using either literature data or experimental measurements — especially for reaction constants, which are always difficult or even impossible to measure in living cells — and thus a large batch of simulations are generally required not only to analyze the dynamical behavior of the system, but also to corroborate the choice of the initial parameters. Therefore, despite the noticeable boost that a fine-grain GPU-based parallelization of stochastic algorithms can have in terms of single simulations, most of the times the effective requirements for the analysis of real models naturally rely upon *coarse-grain* and massively parallel executions of a large number of simulations, as proposed in this work.

Despite all these possible optimizations, stochastic simulations of complex biological systems still remain a computationally intensive task, especially when some molecular species occur in very low amounts and other in very large amounts — or also when the values of reaction constants span over different orders of magnitude — possibly inducing a slowdown of running time because of stiffness and multi-scale problems. Therefore, future efforts will be focused on the implementation of GPU-based hybrid simulation algorithms (see Section 2.3.3): this topic is discussed in Chapter 11.

Chapter 6

Parameter Estimation of biological systems

One of the foremost goals of Systems Biology consists in the development of computational methods to analyze the functioning of cellular systems, whose behavior emerges as a system-level property from the complex interactions of many molecular components and cannot be easily clarified by experimental research only. As described in Chapters 1 and 2, mathematical modeling, simulation techniques and analysis methods represent indispensable tools to describe and better understand the molecular mechanisms of cellular processes.

Nevertheless, the lack of knowledge of physical parameters related to the biochemistry of cellular systems (e.g., reaction rates) poses a limit to the effectiveness of mathematical modeling and *in silico* simulations. These parameters are generally hard or even impossible to measure directly, especially *in vivo*, a significant problem leading to the definition of parameter estimation (PE) problem [59, 276].

The goal of PE is the inference of these unknown values, that is carried out by exploiting the available experimental data related to some biochemical entities (e.g., the intracellular level of some molecular species), which can be measured more easily by standard laboratory protocols. Many methodologies for PE of stochastic systems have been proposed, most of them relying either on some approximation strategy [185, 276, 288], probabilistic methods [224, 264, 349], global optimization [26, 78, 216], or a combination of these approaches [316]. In general, traditional methods based on Gradient Descent are unsuitable for PE because of two factors:

1. the fitness landscape is generally multi-modal, so that the probability of converging to local minima is high, even by exploiting a multi-start strategy [211];

2. in the case of stochastic biological systems, the fitness landscape is rugged, because of stochastic fluctuations, a condition that generates a plethora of local minima which mislead the optimization process.

This chapter follows the line of research of global optimization for PE, and proposes a methodology, called cuPEPSO, characterized by the following features:

- it is based on PSO, the Swarm Intelligence meta-heuristic described in Section 3.3.2;
- it considers as target for the optimization a set of discrete-time measurements, obtained in different experimental conditions and replicated a certain number of times. For this reason, cuPEPSO exploits a multi-swarm version of PSO;
- it considers stochastic models of cellular systems, and relies on an efficient stochastic simulation algorithm (tau-leaping, described in Section 2.3.2) to generate the temporal evolution of the system under investigation [46];
- it exploits GPUs to reduce the computational costs (see Chapter 4 and Section 5.3).

The rationale behind each of these features is explained hereby.

First, in PSO a swarm of candidate solutions (the particles) is randomly generated and undergoes an iterative improvement process that is driven by a fitness function. In cuPEPSO, each particle corresponds to a candidate model parameterization. Through the collective movement of its particles, the swarm is expected to converge to a (global) optimal solution. Albeit PSO does not feature a true convergence theorem like other evolutionary techniques — for instance, the schema theorem proposed in [137] for GAs — it empirically showed a remarkable ability in the optimization of real-valued problems. In particular, PSO was proven to be more suitable for the estimation of parameters of biochemical systems with respect to other methods, as GAs [26, 232], or Covariance Matrix Adaptation Evolution Strategy and Differential Evolution [78].

Second, cuPEPSO was developed with the specific purpose of determining the values of kinetic constants in stochastic RBMs of cellular systems. The framework of cuPEPSO is inspired by the quite common scenario of biology laboratories, where multiple experiments are carried out in different initial conditions (e.g., nutrients, temperature), in order to collect a wide spectrum of information about the functioning of the investigated cellular system in both physiological and perturbed states. This background suggested a multi-swarm version of PSO, where each swarm is assigned to a different experimental condition, and all swarms cooperate to estimate the kinetic values that allows to generate the expected dynamics of the cellular system. The underlying biochemical assumption is that there exists a common set of model parameters that can simultaneously fit all experimental data in the various conditions. This holds if the functioning of the cellular system is not altered by the different experimental settings — that is, it does not rely on other biochemical processes that are not included in the model under investigation and that might be necessary to explain the target data — as it is presumed in the formulation of the problem.

Third, cuPEPSO enables to take into account two relevant factors in the analysis of cellular systems: the role of biological noise and the shortage of experimental sampling. On the one hand, it allows to tame the error due to the stochastic fluctuations that are intrinsic in most cellular pathways, since it relies on the analysis of a large set of simulated dynamics for each candidate model parameterization. The motivation for considering stochastic models of cellular processes comes from several experimental investigations (see, e.g., [31, 85, 90]), which evidenced the role of biological noise in living cells. Similar works highlighted the inadequacy of the deterministic modeling approach — based on ODEs — to describe phenomena such as signaling pathways, especially when small populations of reactant species are involved, therefore supporting the need for stochastic modeling approaches [27, 193, 307, 355]. On the other hand, cuPEPSO deals with sparsely (discrete-time) sampled data consisting in temporal series of molecular species amounts; usually, these measurements are carried out in about ten time instants. Besides, it is considered the case in which the laboratory measurement corresponding to each experimental condition is replicated: usually, this is done up to three times, in order to account for any possible experimental error as well as for the variability of the system behavior. These factors are here taken into account through the definition of a proper fitness function, which is defined as the relative point-to-point distance between the set of available measurements and the simulated dynamics of the corresponding molecular species in the stochastic model. In particular, the average of a number of stochastic simulations is evaluated, run by using the set of parameters encoded by each particle in each swarm and considering the initial condition assigned to the swarm as target. In addition, to let the swarms cooperate in the determination of a model parameterization that fits all experimental conditions, the global best particle of each swarm migrates toward another swarm at regular intervals of iterations, thus sharing the estimates of parameters values of each swarm among all swarms.

Fourth, since cuPEPSO requires a large number of fitness evaluations, which are based on the execution of several stochastic simulations, the whole method was developed for GPUs to reduce the computational costs. This implementation is favored by the fact that all stochastic simulations are mutually independent and can thus be executed in parallel. More specifically, cuPEPSO exploits this parallel model of computation by launching a massive number of GPU threads, so that the stochastic simulations — executed by means of cuTauLeaping [235], the GPU-based implementation of tau-leaping [46] described in Section 5.3 — and the fitness evaluations can be performed concurrently, at each iteration of cuPEPSO.

The main novelty of cuPEPSO consists in the comprehensive integration of these four features that, in previous works, were exploited one at a time for PE or similar tasks. For instance, the whole methodology is based on PSO, since it was shown to be among the most effective optimization techniques [78] and avoids the need for extremely large numbers of simulations that are typical of probabilistic approaches [224, 264]. In contrast to [167], that exploits a fitness function based on multiple target data to discriminate among models with similar behavior, cuPEPSO's multi-swarm methodology takes advantage of a different cooperative approach, under the reasonable assumption that the underlying chemical-physical processes do not change across the different experiments. Finally, no current PE methods rely on tau-leaping to achieve fast simulations of stochastic models, nor exploit a GPU parallelization to speed up the inference process.

Some contents of this chapter are published in [231, 232]. A paper describing the cuPEPSO algorithm presented in this thesis has been submitted to the IEEE Transanctions on Evolutionary Computation [238].

6.1 PE in stochastic models of cellular systems

In this section it is first provided a formalization of the target data exploited by the PE method, which is necessary to describe the fitness function and cuPEPSO's peculiar multi-swarm methodology. Then, the GPU implementation of cuPEPSO is described in detail.

6.1.1 Experimental data and simulated dynamics

As target data for the PE, cuPEPSO takes into account a typical scenario of biological laboratory research, where few replicates of an experiment are usually carried out starting from a set of different initial conditions, each one corresponding to some genetic, chemical or environmental perturbation of the cellular system Ω. Therefore, it is assumed the availability of the following experimental target data:

- A set of measurements corresponding to D different initial conditions, with $D \geq 1$, that we assume to be characterized by distinct initial amounts of some molecular species appearing in Ω.
- For each initial condition, the experimental measurements are replicated E times, for some $E \geq 1$ (usually, $E \leq 3$ in real laboratory experiments). Repetitions allow to account for the errors occurring during the measurement procedures, as well as for the variability of the system's response due to the intrinsic biological noise.
- For each initial condition and for each replicate, it is considered the measurement of the molecular amounts of a subset of species $S_1, \ldots, S_K$, $K \leq N$(where N is the total number of chemical species in Ω), determined by means of standard laboratory technologies. It is assumed hereby that these values correspond to integer numbers of molecules, but this choice is not restrictive, since it is straightforward to transform any real-valued intracellular concentration into the corresponding copy number of molecules, and vice versa [101].
- For each initial condition, for each replicate and for each measured species, the experimental data are assumed to be taken at some time points $t_1, \ldots, t_{T_d}$, not necessarily sampled at regular intervals along the time course of the experiment. It is assumed that the number of sampled time instants might be different from one initial condition to another.

In what follows, $Y_k^{d,e}(t_h)$ denotes the amount of species S_k, measured at time t_h in the replicate e of initial condition d, with $k = 1, \ldots, K$, $h = 1, \ldots, T_d$, $e = 1, \ldots, E$, $d = 1, \ldots, D$, and this set of measures is named a *discrete-time target series* (DTTS). An example of DTTS for species S_k in the experimental condition d, corresponding to three replicates e_1, e_2, e_3, can be seen in Figure 6.1, left graphic.

In order to estimate the values of the reaction constants in $\mathcal{R}$, the DTTS of all measured species must be compared with *in silico* simulations of the dynamics of species in Ω, which are generated by relying on the stochastic model of Ω and exploiting the tau-leaping algorithm (Section 2.3.2).

Namely, given the state of the system Ω at some time instant — specified by the molecular amounts of all the species in $\mathcal{S}$ — and a vector $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_M)$ of

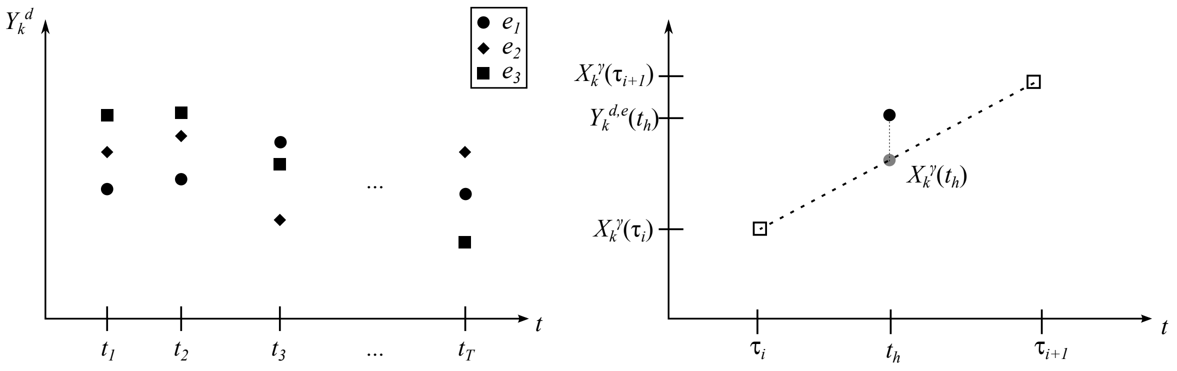


Figure 6.1: *(Top)* Discrete-time target series for species S_k in the experimental condition d, corresponding to three replicates e_1, e_2, e_3 of the same experiment. *(Bottom)* Identification of the interpolated value $X_k^{\gamma,d}(t_h)$, exploiting a single execution of tau-leaping with parameters $\boldsymbol{\gamma}$.

stochastic constants associated to the reactions in $\mathcal{R}$, tau-leaping is used to generate the temporal evolution of Ω. This is achieved by computing, during each iteration, the time interval τ required for the execution of a certain number of reactions sampled from Poisson distributions [46]. By abuse of notation, in what follows τ denotes both the time interval and the left endpoint of the interval itself. The execution of tau-leaping determines a set of consecutive time instants $\tau_0, \dots, \tau_{max}$ — where τ_0 and τ_{max} are the fixed initial and last instants of the simulation — such that at the end of each step of length τ, with $\tau_0 \leq \tau \leq \tau_{max}$, the state of the system is instantly updated by removing (adding) the molecules that appear as reagents (products) in the set of tossed reactions. The length of each time interval τ, as well as the set of tossed reactions, are calculated by exploiting the propensity functions of the reactions (see Equation 2.4).

$X_k^{\gamma,d}(\tau)$ denotes the molecular amount of the target species S_k at time τ in the initial condition d, with $k = 1, \dots, K$, $\tau_0 \leq \tau \leq \tau_{max}$ and $d = 1, \dots, D$, obtained by running tau-leaping with some values $\gamma_1, \dots, \gamma_M$ of the stochastic constants, as specified in an arbitrary vector $\boldsymbol{\gamma}$. In order to compare the simulated amount of species S_k obtained by tau-leaping, and the measured amount of the same species as given in the DTTS, it is necessary to determine the amount of S_k taken in correspondence to each experimentally sampled time point t_h, $h = 1, \dots, T_d$, hereby denoted by $X_k^{\gamma,d}(t_h)$. This can be done, for each sampled instant t_h, by choosing the two consecutive time instants $\tau_i, \tau_{i+1} \in [\tau_0, \tau_{max}]$ in a tau-leaping simulation, such that $\tau_i \leq t_h \leq \tau_{i+1}$ and there exist no other τ', τ'' with $\tau_i \leq \tau' \leq t_h \leq \tau'' \leq \tau_{i+1}$. Then, by linear interpolation between the values $X_k^{\gamma,d}(\tau_i)$ and $X_k^{\gamma,d}(\tau_{i+1})$, the value $X_k^{\gamma,d}(t_h)$ is computed (see Figure 6.1, right graphic).

6.1.2 The fitness function

In cuPEPSO, the fitness function of each candidate parameters vector $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_M)$ is evaluated by comparing the measured DTTS with the outcome of a number of stochastic simulations, each one performed by using the values of the putative reaction constants specified in $\boldsymbol{\gamma}$.

Specifically, given an initial condition d, for each vector $\boldsymbol{\gamma}$ a number G of independent simulations is executed, which are run in parallel by using the GPU-powered simulator cuTauLeaping [235] described in Section 5.3. Then, for each target species S_k it is determined how much its average simulated molecular amount at time instant t_h — denoted as $\langle X_k \rangle_G^{\gamma,d}(t_h)$ and calculated according to the outcome $X_k^{\gamma,d}(t_h)$ of the G parallel simulations — differs from the experimental measures $Y_k^{d,e}(t_h)$ of the DTTS at the corresponding time instant and in the same experimental condition d. To this aim, the point-to-point distance between $\langle X_k \rangle_G^{\gamma,d}(t_h)$ and $Y_k^{d,e}(t_h)$ is measured, averaging over all E replicates carried out in condition d and over all numbers T_d of sampled time instants.

Thus, the fitness function is defined as

$$\mathcal{F}_d(\boldsymbol{\gamma}) = \frac{1}{ET_d} \sum_{h=1}^{T_d} \sum_{k=1}^{K} \sum_{e=1}^{E} f(Y_k^{d,e}(t_h), \langle X_k \rangle_G^{\gamma,d}(t_h)), \tag{6.1}$$

where:

- if $Y_k^{d,e}(t_h) > 0$, then

$$f(Y_k^{d,e}(t_h), \langle X_k \rangle_G^{\gamma,d}(t_h)) = \frac{|Y_k^{d,e}(t_h) - \langle X_k \rangle_G^{\gamma,d}(t_h)|}{Y_k^{d,e}(t_h)},$$

  that is, the normalized point-to-point distance between the average simulated amount of S_k and the corresponding DTTS is calculated;

- if $Y_k^{d,e}(t_h) = 0$, then

$$f(Y_k^{d,e}(t_h), \langle X_k \rangle_G^{\gamma,d}(t_h)) = \langle X_k \rangle_G^{\gamma,d}(t_h),$$

  that is, if the experimentally measured value of S_k is zero, only the average simulated amount of S_k is taken into account.

The motivation for averaging over the number of sampled time instants, T_d, is that different experimental conditions might be characterized by a different set of samples.

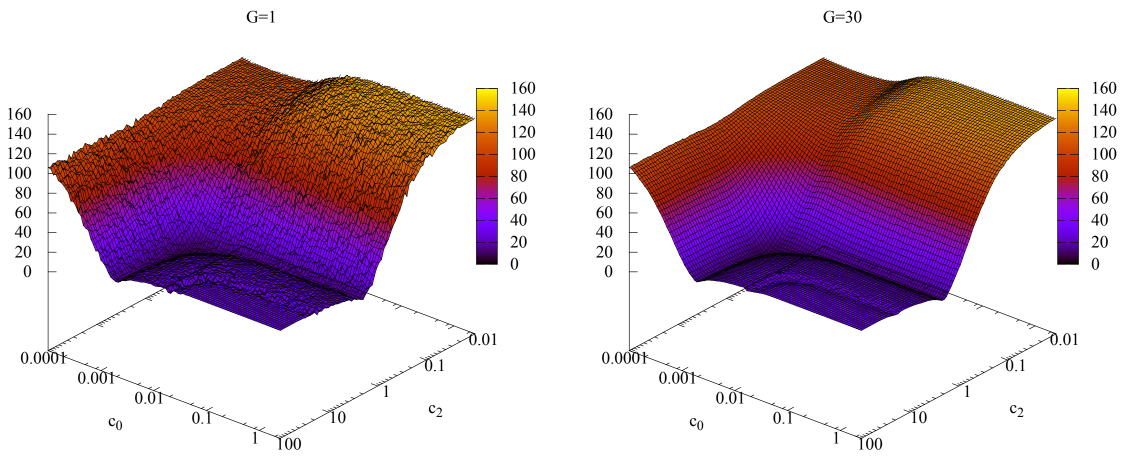


Figure 6.2: Comparison of the fitness landscape of the MM model projected on the 2D-space corresponding to parameters c_0 and c_2, using $G = 1$ (*left*) and $G = 30$ (*right*) stochastic simulations. The averaging of the dynamics smooths the landscape, that indeed appears more rugged if $G = 1$ because of the effect of stochastic fluctuations in the molecular species amounts.

The average reduces any bias in the choice of the best candidate parameters vectors in the multi-swarm version of PSO, as described in Section 6.1.3.

The rationale behind the use of the average of G tau-leaping executions is that stochastic fluctuations are usually present in the simulated dynamics of molecular species, especially when the number of molecules in the cellular system is in the order of a few units or tens. Since each stochastic simulation — performed with tau-leaping algorithm using the same parameterization $\boldsymbol{\gamma}$ — determines a different realization of the probability distribution of reactions propensity functions, thanks to the calculation of an average dynamics the effect of stochastic fluctuations is mitigated and a more precise fitness function value can be computed. As a consequence, smoothing down the roughness of the fitness landscape can improve the effectiveness of the PE method. An example of this issue is shown in Figure 6.2, where the fitness landscape of a simple enzymatic kinetics, the MM model (see Appendix A.1 for a formal definition), is plotted for $G = 1$ (left graphic) and $G = 30$ (right graphic).

The fitness function given in Equation 6.1 has two characteristics:

- Since the goal is to identify a model parameterization $\boldsymbol{\gamma}$ whose simulated dynamics overlaps the DTTS, the PE problem in cuPEPSO can be stated as a minimization problem. Anyway, since cuPEPSO exploits a stochastic algorithm to generate the temporal evolution of the molecular species, it is generally unlike to reach the (ideal) value of zero for the fitness calculation, due to the intrinsic stochasticity of the system.

- The value $\mathcal{F}_d(\boldsymbol{\gamma})$ determines the quality of an arbitrary parameters vector $\boldsymbol{\gamma}$ with respect to the DTTS measured in the experimental setting d, independently from the other initial conditions. In Section 6.1.3, it is shown how to exploit a multi-swarm PSO to estimate a common set of parameters $\boldsymbol{\gamma}$ that can simultaneously fit all available DTTS in all initial conditions. Stated otherwise, in cuPEPSO it is considered a multi-swarm architecture of PSO to deal with multiple fitness-cases — distributing one case per swarm — and particles migration is exploited to achieve a unique result for all these cases.

6.1.3 A multi-swarm structure for the PE problem

The multi-swarm topology of PSO is structured as follows. For each initial condition d=1, . . . , D it is considered a swarm σ_d, consisting of n particles $\boldsymbol{\gamma}_d = (\gamma_{d,1}, \ldots, \gamma_{d,M})$, whose components correspond to the values of the stochastic constants of reactions in $\mathcal{R}$, with $\gamma_{d,\mu} \in \mathbb{R}^+$, $\mu = 1, \ldots, M$. As described in Section 6.1.2, each vector $\boldsymbol{\gamma}_d$ is used to execute G simulations with tau-leaping in order to generate, for all target species S_k, the sets of molecular amounts $\langle X_k \rangle_G^{\gamma;d}(t_h)$ associated to particle $\boldsymbol{\gamma}_d$. The fitness of particle $\boldsymbol{\gamma}_d$ is then evaluated according to Equation 6.1, using the experimental data $Y_k^{d,e}(t_h)$ corresponding to the experimental condition d. So doing, each swarm performs the estimation of stochastic constants independently from the other swarms and determines, for each iteration IT, its global best particle $\boldsymbol{\gamma}_d^{best}$, which is the parameters vector that matches all replicates of the DTTS in condition d in the best possible way.

Afterwards, in order to estimate a set of stochastic constants that is common to all swarms, and that is able to reproduce the expected system dynamics in all experimental conditions, particles are allowed to migrate among swarms. The migration takes place at regular intervals, every κIT_{mig} iterations, with $1 \leq \kappa \leq \lfloor IT_{\texttt{MAX}}/IT_{mig} \rfloor$ and $1 \leq IT_{mig} \leq IT_{\texttt{MAX}}$, where $IT_{\texttt{MAX}}$ is the maximum number of iterations of the PSO. To better explain migration, the topology of the multi-swarm PSO can be formalized as a directed graph $\mathcal{G} = \langle V, A \rangle$, where $V = \{\sigma_1, \ldots, \sigma_D\}$ is the set of D vertices (the swarms) and $A = \{(\sigma_{d'}, \sigma_{d''}) \mid \text{particles can migrate from swarm } \sigma_{d'} \text{ to swarm } \sigma_{d''}\}$ is the set of edges. It is also assumed that, considering any edge in A, exactly one particle migrates from $\sigma_{d'}$ to $\sigma_{d''}$, in order to assure that the size n of each swarm is not altered by migration.

Different interconnection topologies among the swarms can be defined, for instance the one proposed in [232] or [284]. Here, a *dynamic topology* (DT) is considered, since it was shown in [232] to lead to better results with respect to a *static topology* (ST).

The DT is characterized by a set of edges $A_{dyn} \subseteq A$, with $|A_{dyn}| = D$, that is generated with a random permutation on the ordering of the swarms and updated at each step of migration. Note that, so doing, also an edge $(\sigma_{d'}, \sigma_{d'})$ can belong to A_{dyn}, which means that swarm $\sigma_{d'}$ is isolated and will neither receive nor send particles to other swarms at that step of migration. Then, the migration acts as follows: for each edge $(\sigma_{d'}, \sigma_{d''})$ in A_{dyn} (including the case $\sigma_{d''} = \sigma_{d'}$), at the IT_{mig}-th iteration the worst particle of swarm $\sigma_{d''}$ is deleted and replaced by the global best particle $\gamma_{d'}^{best}$ of swarm $\sigma_{d'}$.

A similar notion of DT for migration was introduced in [187] for parallel GAs, although in that work the migration topology was dynamically updated according to some properties of the population. Fernández *et al.* also analyzed ST and DT (named, respectively, *ring* and *random* topologies) in the case of multi-island Distributed Genetic Programming, showing that no specific topology is always better than the others, even though DT is computationally more efficient and more effective in the case of small populations [91].

The multi-swarm PSO exploited in this chapter resembles the island-model of EC described in [328, 333], where a population of candidate solutions is partitioned into a set of disjoint sub-populations, which evolve independently but interact by means of periodic migrations. In PSO terms, each sub-population corresponds to a swarm and the migration process is represented by the movement of particles from one swarm to another, a formulation that is similar to the multi-swarm PSO described in [341].

6.1.4 GPU implementation of cuPEPSO

In order to complete the PE process, cuPEPSO has to evaluate the fitness function $\mathcal{F}_d$ given in Equation 6.1 for all swarms $d = 1, \dots, D$. This corresponds to the execution of $D \times G \times n \times IT_{\texttt{MAX}}$ simulations for each optimization run, which can be in the order of millions, and therefore would result computationally expensive. However, since PSO is an inherently parallel algorithm, the fitness of each particle can be evaluated independently from the other particles. Among the existing parallel architectures, cuPEPSO was developed to exploit the GPGPU platform. Specifically, cuPEPSO is implemented to exploit Nvidia's CUDA (see Chapter 4.1 for further information).

For each initial condition $d = 1, \dots, D$, exactly G blocks (corresponding to the G parallel tau-leaping simulations) are created, thus launching a total of $D \times G$ blocks; every block is composed of n threads. $\mathcal{B}_{d,g}$ denotes a block belonging to the d-th swarm, associated to the d-th initial condition, whose threads are performing the g-th parallel simulation. $\mathcal{T}_{i,d,g}$ denotes the i-th thread belonging to block $\mathcal{B}_{d,g}$, i.e., the thread associated to the i-th particle in the d-th swarm that performs the g-th parallel simulation (with $i = 1, \dots, n, \ \ d = 1, \dots, D, \ \ g = 1, \dots, G$). Each thread $\mathcal{T}_{i,d,g}$ executes a parallel tau-leaping simulation using the vector of parameters $\boldsymbol{\gamma}$ of the i-th particle of swarm σ_d and exploiting cuTauLeaping [235], the GPU-accelerated version of tau-leaping presented in Section 5.3. So doing, $\mathcal{T}_{i,d,g}$ generates the molecular amounts $X_k^{\gamma,d}(t_h)$, for $t_h = 1, \dots, T_d$, for the g-th parallel repetition. The G parallel repetitions are collected and used to calculate the average dynamics $\langle X_k \rangle_G^{\gamma,d}(t_h)$, for $t_h = 1, \dots, T_d$, exploited by the fitness function shown in Equation 6.1. The average dynamics, and the fitness evaluations, are again performed in parallel by $D \times n$ threads, distributed in D blocks.

As cuPEPSO relies on a synchronous implementation of PSO, the process of optimization starts as soon as cuTauLeaping's execution is completed. Each simulation halts as it reaches the last sampled time instant T_d. Finally, every IT_{mig} iterations of the PSO the migration of particles occurs, as described in Section 6.1.3.

6.2 Results

This section presents some results of the PE performed by cuPEPSO on two different stochastic models. For each model, the tests were performed with two steps: first, the best settings of cuPEPSO with the specific model are identified, by analyzing their impact on the estimation performances; second, the PE is performed using the identified best settings. Finally, this section provides a brief comment about the computational performances of cuPEPSO.

6.2.1 Stochastic models

Two stochastic RBMs were considered to test the effectiveness of cuPEPSO, the MM model and the PGN model. The set of chemical reactions corresponding to MM and the values of their associated stochastic constants are given in Appendix A.1. All species of the MM model (S, E, P, and ES) were considered as target to perform the estimation of the stochastic constants. In the PE process $D = 4$ distinct experimental conditions were assumed, and $E = 3$ replicates for each condition, which are characterized by the following initial amounts of substrate and enzyme species:

- $S = 1000$, $E = 750$ molecules, associated to swarm σ_1;
- $S = 2000$, $E = 750$ molecules, associated to swarm σ_2;
- $S = 500$, $E = 750$ molecules, associated to swarm σ_3;

- $S = 1000$, $E = 500$ molecules, associated to swarm σ_4.

The initial amounts of the remaining species are zero.

The set of chemical reactions corresponding to PGN and the values of their associated stochastic constants are given in Appendix A.2. All species of the PGN model (DNA, P, P_2, DNA:P_2, and $mRNA$) were considered as target to perform the estimation of the stochastic constants. In the PE process $D = 4$ distinct experimental conditions were assumed, and $E = 3$ replicates for each condition, which are characterized by the following initial amounts of DNA:

- $DNA = 50$ molecules, associated to swarm σ_1;
- $DNA = 100$ molecules, associated to swarm σ_2;
- $DNA = 250$ molecules, associated to swarm σ_3;
- $DNA = 500$ molecules, associated to swarm σ_4.

The initial amounts of the remaining species are zero.

For both models, the target DTTS were generated *in silico* by averaging the molecular amounts of target species over 1000 stochastic simulations, and then sampling the outcome at $T_d = 10$ time instants for each initial condition and each replicate. These stochastic simulations were executed by using the values of stochastic constants given in Tables A.1 and A.3.

6.2.2 PE methodology analysis

To analyze the performances and to investigate the effectiveness of the PE methodology presented in this thesis, it is assumed that a good solution is identifiable by relying only on its fitness value. Indeed, the values of stochastic constants in the vector $\boldsymbol{\gamma}_c$ cannot be used to establish the goodness of a candidate model parameterization because of two reasons. First, the values of stochastic constants are usually not available. Second, there can be several sets of stochastic constants values (different from $\boldsymbol{\gamma}_c$) that can equally fit the DTTS. Therefore, the *best* solution will correspond to the particle characterized by the lowest fitness value among all swarms, found by cuPEPSO at the last PSO iteration.

Let $\boldsymbol{\gamma}_d^{best}(IT)$ be the best particle of swarm σ_d found at the IT-th iteration of the optimization process, for some $IT \leq IT_{\mathtt{MAX}}$. The *multi-swarm best particle* at the IT-th iteration, denoted by $\boldsymbol{\gamma}^{\#}(IT)$, is defined as the best particle found at the

IT-th iteration that is characterized by the minimum fitness value across all swarms. Formally,

$$\boldsymbol{\gamma}^{\#}(IT) = \boldsymbol{\gamma}_d^{best}(IT)$$

such that $\mathcal{F}_d(\boldsymbol{\gamma}_d^{best}(IT)) = \min_{d=1,\ldots,D}\{\mathcal{F}_d(\boldsymbol{\gamma}_d^{best}(IT))\}$.

Then, the best model parameterization of the whole PE process, denoted by $\boldsymbol{\gamma}^*$, is defined as the multi-swarm best particle found at the last iteration, that is,

$$\boldsymbol{\gamma}^* = \boldsymbol{\gamma}^{\#}(IT_{\mathtt{MAX}}).$$

Particle $\boldsymbol{\gamma}^*$ contains the values of the stochastic constants that will be used to generate the simulated dynamics of the model and to do the comparison with the available DTTS, in all initial conditions.

Several preliminary tests were executed, to determine the influence of the settings of cuPEPSO on the PE process. To this aim, an appropriate measure had to be defined to compare the convergence speed and the quality of the multi-swarm best particles determined in each tested cuPEPSO setting. To this aim, the *Average Best Fitness* (ABF) at each iteration IT is defined as the mean of the fitness value of the multi-swarm best particles found at iteration IT, evaluated over a number ϱ of cuPEPSO runs. Formally,

$$\text{ABF} = \frac{1}{\varrho}\sum_{j=1}^{\varrho}\mathcal{F}_d(\boldsymbol{\gamma}_j^{\#}(IT)),$$

where $\boldsymbol{\gamma}_j^{\#}(IT)$ denotes the multi-swarm best particle found at the j-th run of cuPEPSO (the indication of the IT-th iteration is omitted from the notation of ABF for succinctness).

To determine the best setting for PE, cuPEPSO was applied on the MM and PGN models by varying a single cuPEPSO parameter at a time (if not otherwise specified), trying to determine its optimal value for each biological model. These tests are also useful to investigate whether the optimal cuPEPSO settings change according to the characteristics of model, or else some general configuration can be determined. Besides the parameter being modified, all tests share the following common cuPEPSO setting:

- swarms size $n = 64$ particles;
- logarithmic sampling of particles initial position in $[1 \cdot 10^{-4}, 1 \cdot 10^{1}]$;
- inertia weight w linearly decrementing from 0.9 to 0.4;

- cognitive factor $C_{cog} = 1.9$;
- social factor $C_{soc} = 1.9$;
- maximum velocity of particles $v_{max} = 1/5$, that is, $|v_i| \leq 1/5 \cdot (\beta_\mu^{max} - \beta_\mu^{min})$, for $\mu = 1, \ldots, M$;
- damping boundary conditions;
- DT for particle migration;
- migration interval $IT_{mig} = 20$;
- number of iterations $IT_{\texttt{MAX}} = 100$;
- $G = 10$ parallel stochastic simulations for each particle;
- $\varrho = 30$ runs of PE for each test.

In cuPEPSO, particles are initialized using a logarithmic sampling in order to uniformly span over the different orders of magnitude of the search space.

cuPEPSO test on MM model

In the first test, different PE were performed varying the number of particles in each swarm ($n = 16, 32, 48, 64$), to determine how the size of the populations affects the estimation performances. Figure 6.3 shows that, for this model, more than 48 particles does not help the convergence, considering both the convergence speed and the ABF of the multi-swarm best particles. Nevertheless, as described in Section 6.1.4, the highest parallelism of the SMXs on the GPU is achieved by exploiting a number of threads per block (i.e., particles per swarm) that is proportional to a full warp (i.e., 32 threads). Thus, even though 48 particles would be a enough for a good optimization of biological models of this complexity, $n = 64$ is considered as the optimal choice for the swarms size.

The second test consisted in evaluating different values of the number of parallel simulations ($G = 1, 10, 20, 30, 40$), which are used to compute the average amount of molecular species for the fitness calculation (Equation 6.1). Figure 6.4 shows that the best option is $G \geq 30$. This result can be confirmed by a visual comparison of the fitness landscapes of the MM model using $G = 1$ (Figure 6.2, left) and $G = 30$ (Figure 6.2, right). These figures show the fitness values on a 2D-space projection of the fitness landscape, evaluated by varying the stochastic constant of the first reaction

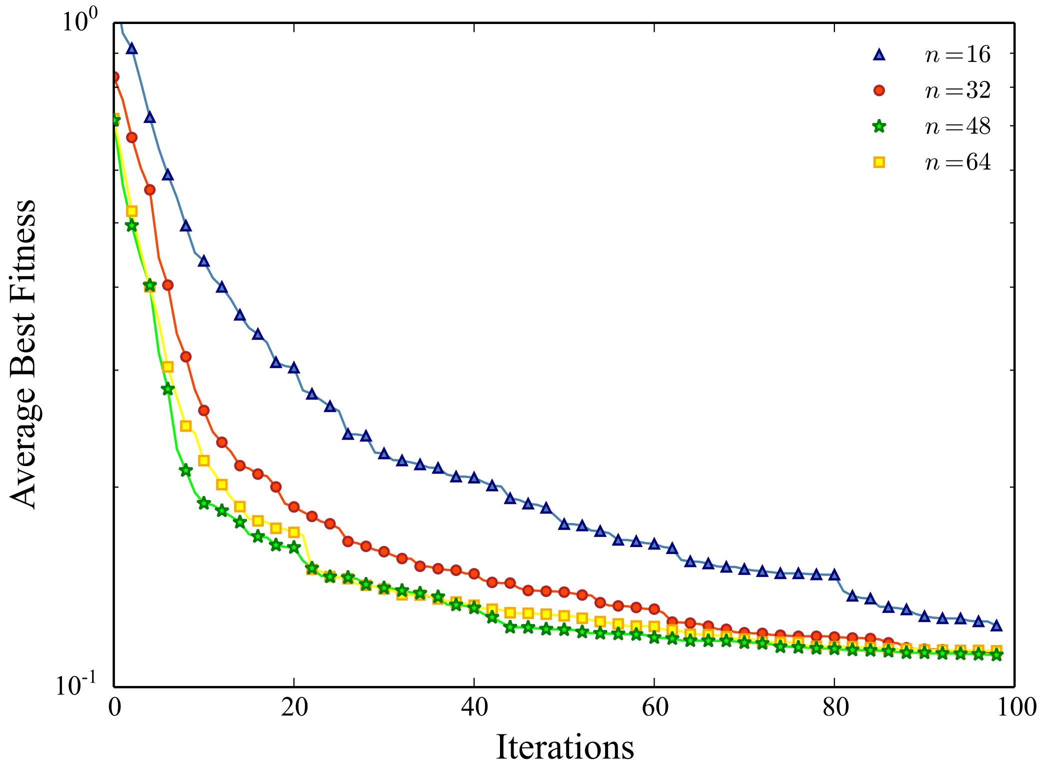


Figure 6.3: ABF of the best particles in the multi-swarm PSO, using different values of the swarm size n to estimate the stochastic constants of the MM model. Results show that $n = 48$ achieves both fast convergence and good estimation.

(the formation of the substrate-enzyme complex) and of the third reaction (the release of the enzyme and synthesis of the final product). The ruggedness of the landscape — corresponding to local minima in the fitness function, which mislead particles and slow down the convergence — is smoothed by high values of G.

In the third test different migration intervals ($IT_{mig} = 1, 5, 10, 20$) were analyzed for the dynamic topology. According to the obtained results, no specific interval yields a result that could be considered better than the others. Therefore, the value $IT_{mig} = 20$ was chosen for the optimal cuPEPSO setting.

The fourth test was focused on the two most relevant parameters for the proper functioning of PSO, namely, the cognitive and social factors C_{cog} and C_{soc}, which regulate the exploitation and exploration capabilities of the algorithm. In particular, the scope of the test was to characterize the synergistic effect of the variation of both factors together. To this aim, the value of both C_{cog} and C_{soc} were varied in the interval $[1, 3]$. The heatmap in Figure 6.5 (top) shows the ABF for the MM model, according to multiple combinations of values for C_{cog} (x-axis) and C_{soc} (y-axis). Darker (lighter)

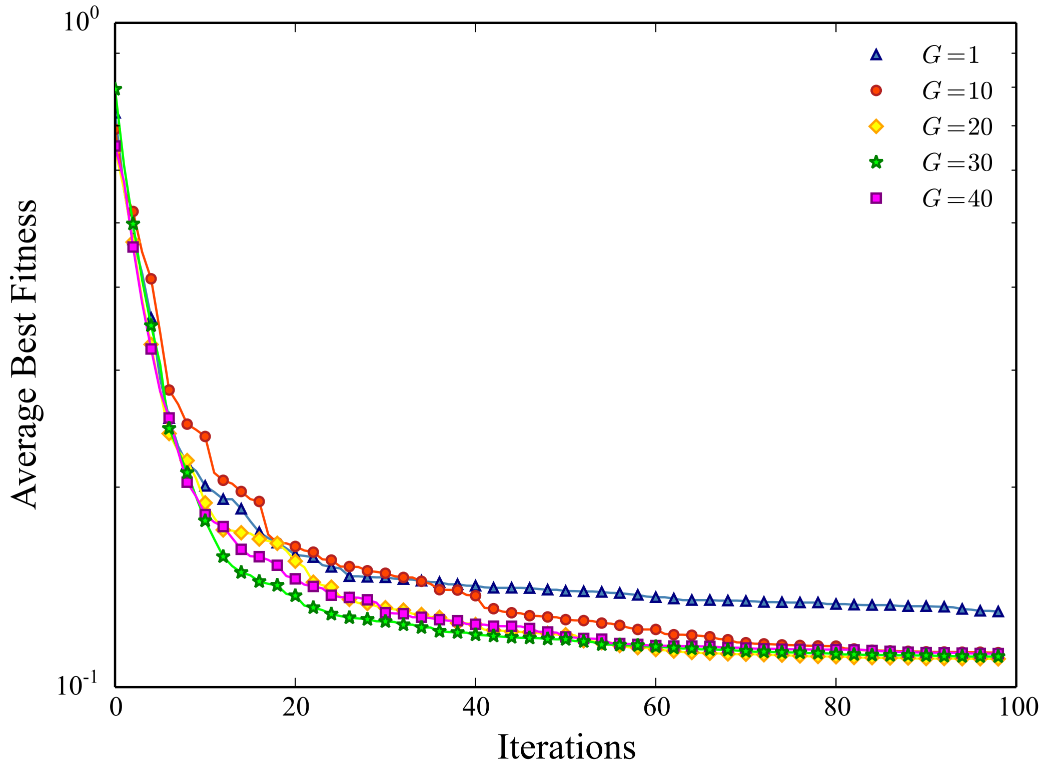


Figure 6.4: ABF of the best particles in the multi-swarm PSO, using different values of the number of parallel stochastic simulations G to estimate the stochastic constants of the MM model. Results show that for $G = 30$ both fast convergence and good estimation are achieved.

colors indicate a lower (higher) ABF, which corresponds to a better (worse) match between the DTTS and the simulated dynamics (generated using $\boldsymbol{\gamma}^*$). Even though the heatmap seems to highlight regions with a better convergence than others (e.g., for the couple $C_{cog} = 2, C_{soc} = 1$), it must be noted that the ABF spans in a very reduced interval of values (from 0.215 to 0.245). As a matter of fact, in the case of the MM model, all the tested values of C_{cog} and C_{soc} allow to converge to an optimal solution. Thus, for extremely simple models as MM, any choice of these factors seems to lead to good solutions.

In the last test, different values for the maximum velocity were considered (namely, $v_{max} = 1/1000, 1/100, 1/10, 1/4, 1/2$). According to the obtained results, for the MM model $v_{max} = 1/2$ is the best option, allowing both faster convergence and better ABF (Figure 6.6). For this model, v_{max} values lower than $1/2$ are not helpful to converge to better results and, in the case of $v_{max} = 1/1000$, the multi-swarm does not even converge to any optimal result before the halting criterion is met. Interestingly, higher values of v_{max} seem to yield a more pronounced scattering of the best solutions in the

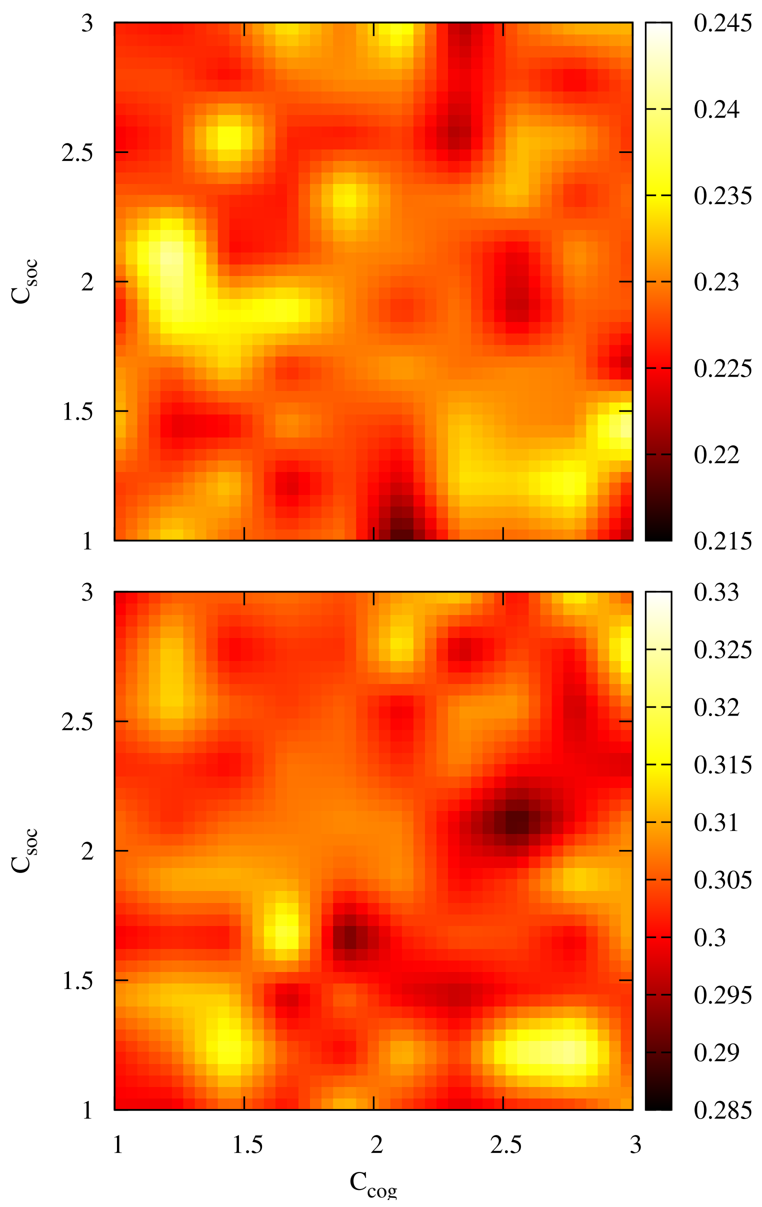


Figure 6.5: Heatmap of the impact of the social and cognitive factors on the estimation performances for the MM model (*top*) and the PGN model (*bottom*). The figure shows the ABF of the best particle at iteration $IT_{\texttt{MAX}}$, by using different values for C_{cog} (x-axis) and C_{soc} (y-axis) chosen in the interval $[1, 3]$. Darker colors indicate a lower ABF value, which corresponds to a better match between the DTTS and the simulated dynamics. In the case of the MM model, the different values of C_{cog} and C_{soc} do not seem to have a real impact on the convergence, since the ABF spans in a very small range of values. On the contrary, in the case of the PGN model, the best solutions are found using the combination $C_{cog} = 2.5$ and $C_{soc} = 2.1$, while the worst results are obtained using a low values of the social factor and a high value of the cognitive factor, corresponding to a local exploitation of particles around the initial random positions.

search space. Figure 6.7 shows the position at iteration $IT_{\texttt{MAX}}$ of the best particle found by each swarm, during each of the ϱ runs of the PE on the MM model. In the case of low values of v_{max} (e.g., $v_{max} = 0.001$), particles are clustered around the global optimum; as the value of v_{max} increases (e.g., $v_{max} = 0.5$), particles are more and more distributed along the axis of the less sensitive parameter for the MM model (i.e., c_1). This phenomenon can be explained by considering that high values of the maximum velocity allow a larger exploration of the variation interval of constant c_1. Being the less sensitive parameter of the MM model, any choice of the c_1 value yields a reduced impact on the simulated dynamics and, consequently, on the evaluation of the fitness values.

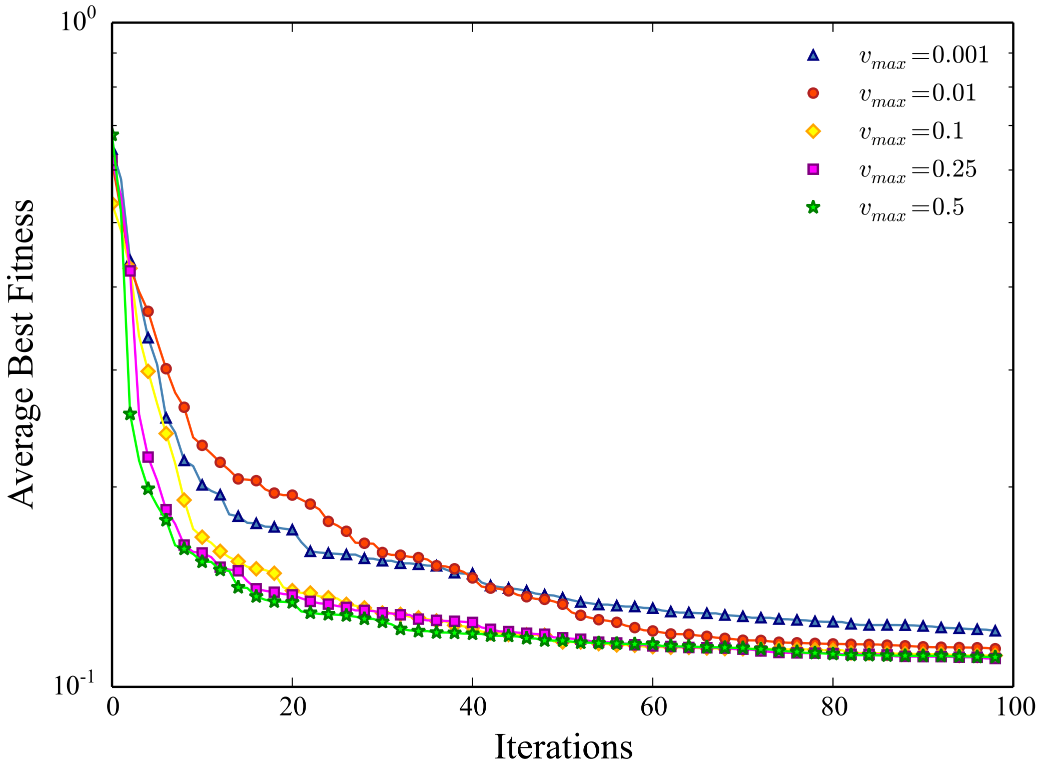


Figure 6.6: ABF of the best particles in the multi-swarm PSO, using different values of the maximum velocity v_{max} to estimate the stochastic constants of the MM model. Results show that $v_{max} = 1/2$ allows both fast convergence and good estimation, whilst the convergence of cuPEPSO is slowed down using very low values (e.g., $v_{max} = 1/1000$).

These preliminary tests allowed the definition of the following optimal cuPEPSO settings for the MM model:

- swarms size $n = 64$ particles;

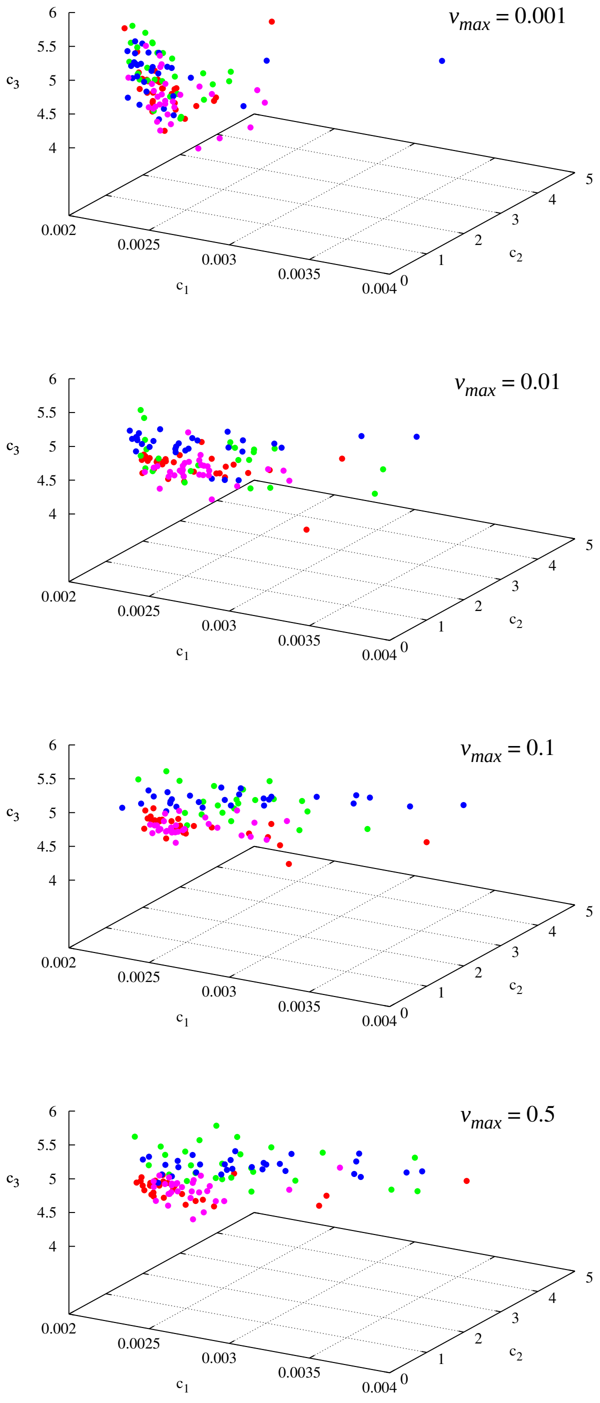


Figure 6.7: Positions of the best particles of each swarm at iteration $IT_{\texttt{MAX}}$ during each run of the PE executed on the MM model, using different values for v_{max}. For higher values of the maximum velocity, the position of the best particles are more scattered in the search space, suggesting worse PE performances.

- logarithmic sampling of particles initial position in $[1 \cdot 10^{-4},\, 1 \cdot 10^{1}]$.
- inertia weight w linearly decrementing from 0.9 to 0.4;
- cognitive factor $C_{cog} = 1.9$;
- social factor $C_{soc} = 1.9$;
- maximum velocity of particles $v_{max} = 1/2$;
- damping boundary conditions;
- dynamic topology for particle migration;
- migration interval $IT_{mig} = 20$;
- $G = 30$ parallel stochastic simulations for each particle.

Figure 6.8 shows that the simulated dynamics of the MM model, generated with the stochastic constants of particle $\boldsymbol{\gamma}^* = (0.0026, 0.6562, 5.2432)$, estimated using the optimal settings of cuPEPSO (with $IT_{\mathtt{MAX}} = 100$), perfectly fits the DTTS. It is worth noting that the multi-swarm best particle is able to fit *all* the DTTS of target species in *all* initial conditions.

cuPEPSO test on PGN model

All tests performed on the MM model were repeated on a the PGN model, in order to investigate whether different models might require different settings of cuPEPSO. The best settings identified for the swarm size n (Figure 6.9) and the migration interval IT_{mig} (data not shown) for the PGN model, are equivalent to the MM model.

Differently from the MM model, the analysis of cognitive and social factors highlighted a more peculiar behavior, shown in the heatmap of Figure 6.5 (bottom). According to results, for the PGN model the best solutions are found using the combination of values $C_{cog} = 2.5$ and $C_{soc} = 2.1$, which are close to similar results previously obtained for the PE problem [78]. The worst results were obtained when the social factor is low and the cognitive factor is high, which correspond to a local exploitation of particles around the initial random positions. This may be explained by the fact that, due to a low attraction towards the global best of each swarm, the exploration capabilities are strongly reduced, therefore also decreasing the quality of the final solution.

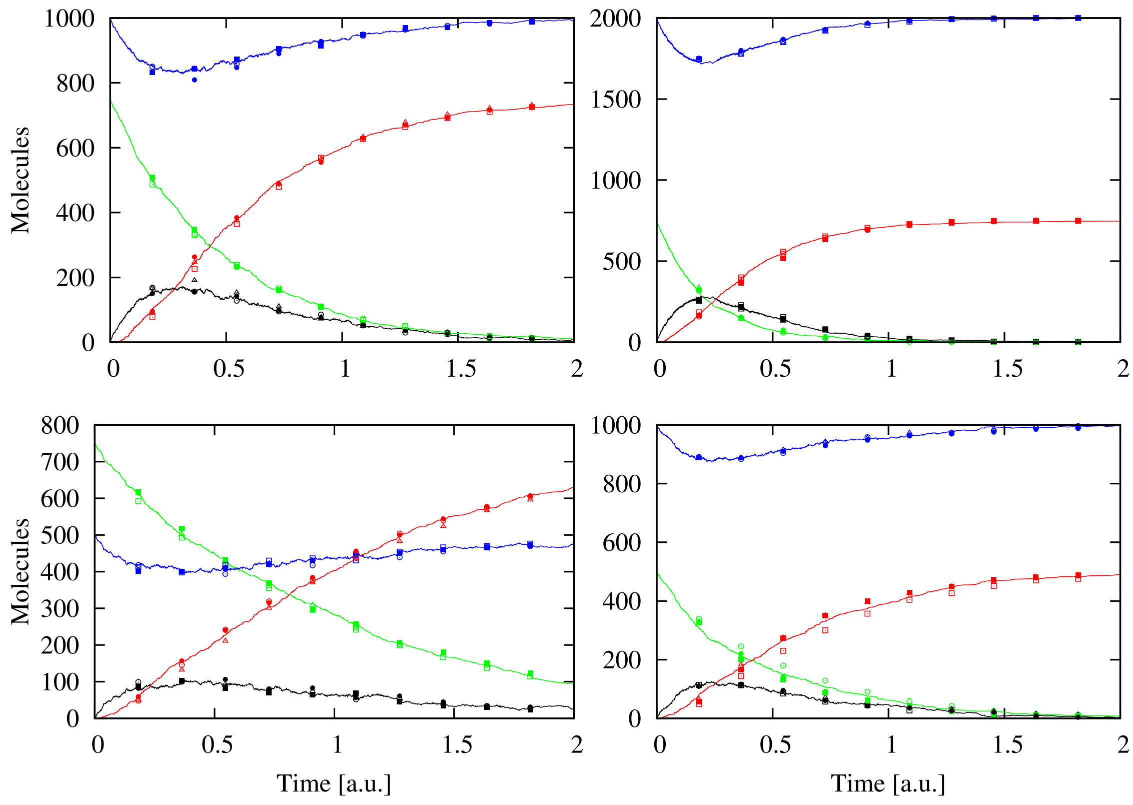


Figure 6.8: Comparison of DTTS (dots) with the simulated dynamics (solid lines) of species S (blue), E (green), ES (black) and P (red) of the MM model. The dynamics were generated with tau-leaping using the stochastic constants of the multi-swarm best particle, and considering the different initial conditions characterizing each swarm. From top to bottom, left to right, the results in swarm σ_1, σ_2, σ_3, σ_4 are shown.

Interestingly, for this model, an average of $G = 10$ simulations is enough to smooth the dynamics and ensure a good convergence (Figure 6.10). Figure 6.11 shows that the best choice for the maximum velocity is $v_{max} = 1/100$, which is completely different from the case of the MM model. Arguably, the choice of G and v_{max} should be selected according to the characteristics of the biological model under investigation, especially for what concerns the ruggedness of the fitness landscape caused by an high variance in the simulated dynamics.

These tests allowed the definition of the following optimal cuPEPSO settings for the PGN model, that were used to perform the PE:

- swarms size $n = 64$ particles;
- logarithmic sampling of particles initial position in $[1 \cdot 10^{-4}, 1 \cdot 10^{1}]$.
- inertia weight w linearly decrementing from 0.9 to 0.4;

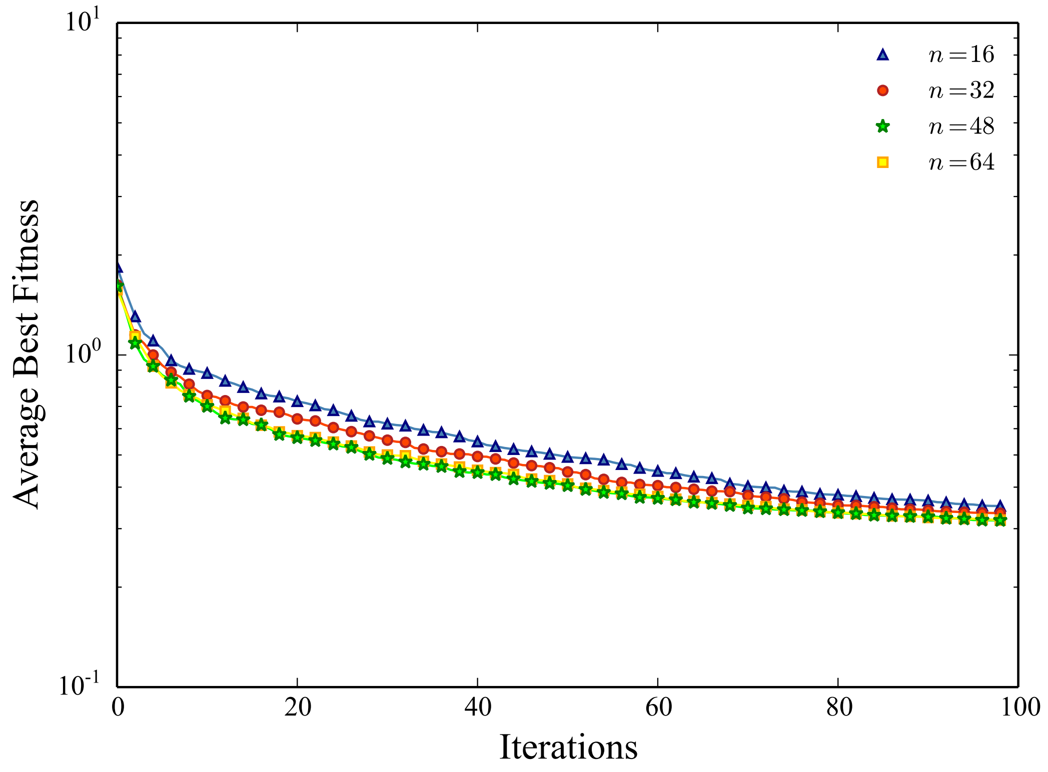


Figure 6.9: ABF of the best particles in the multi-swarm PSO, using different values of the swarm size n to estimate the stochastic constants of the PGN model. Similarly to the case of the MM model, results show that $n = 48$ allows both fast convergence and good estimation.

- cognitive factor $C_{cog} = 2.5$;
- social factor $C_{soc} = 2.1$;
- maximum velocity of particles $v_{max} = 1/100$;
- damping boundary conditions;
- dynamic topology for particle migration;
- migration interval $IT_{mig} = 20$;
- $G = 10$ parallel stochastic simulations for each particle.

Figure 6.12 shows that the simulated dynamics of the PGN model, generated with the stochastic constants of particle $\boldsymbol{\gamma}^* = (0.1344, 0.9706, 0.3646, 0.3236, 0.0910, 0.7448, 0.1947, 0.0647)$, estimated using the optimal settings of cuPEPSO (with $IT_{\texttt{MAX}} = 100$). It is worth noting the multi-swarm best particle is able to perfectly fit *all* the DTTS of target species in *all* initial conditions.

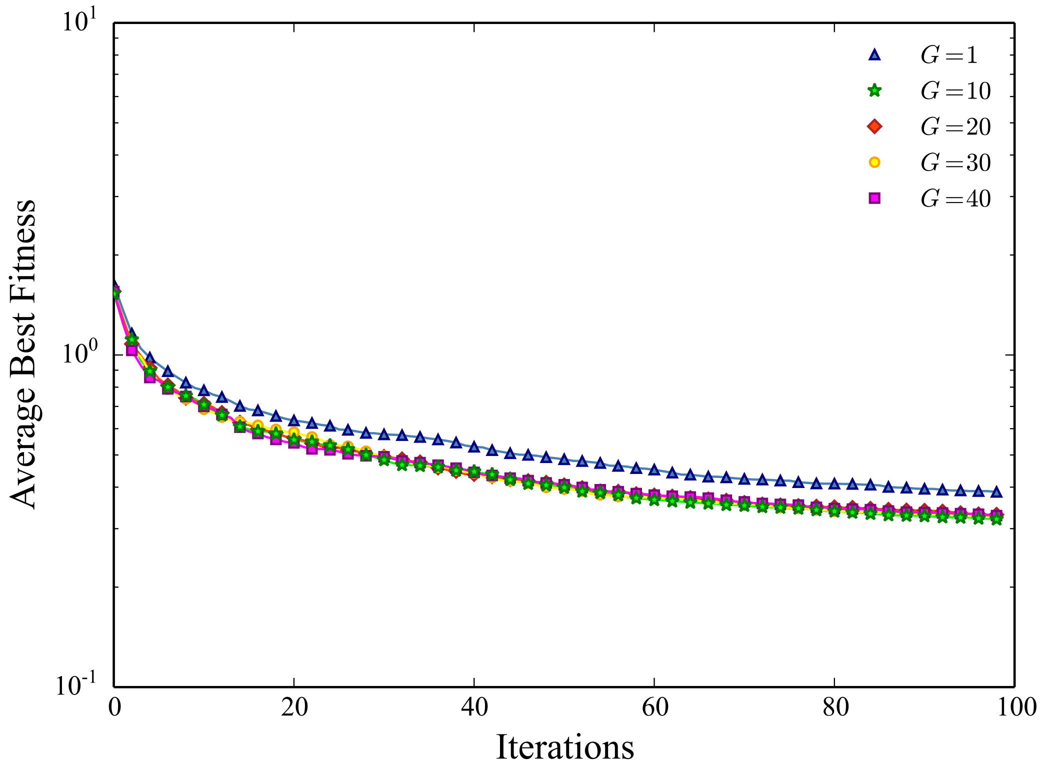


Figure 6.10: ABF of the best particles in the multi-swarm PSO, using different values of the number of parallel stochastic simulations G to estimate the stochastic constants of the PGN model. Results show that, for this model, a choice of $G = 10$ parallel simulations is enough to mitigate the stochastic fluctuations of molecular species amounts, smoothing down the fitness landscape and helping the convergence.

6.2.3 Computational results

cuPEPSO has a relevant computational complexity, both in space and time: the multi-swarm model — in which every particle performs multiple simulations to calculate the average dynamic behavior of a set of species — requires a large amount of memory[1] and a number of computation steps proportional to $O(D \times G \times n \times IT_{\mathtt{MAX}})$ for each PE process. For instance, in the case of the best setting of the MM model, the number of simulations is equal to $64 \times 4 \times 30 \times 100 = 768000$.

Thanks to the GPU acceleration of cuTauLeaping [235], anyway, both n and G can be set to relatively high values without any impact on the overall running time: all simulations are spread across multiple threads and run in parallel on the available SMXs. Hence, fitness evaluations take roughly as much time as a single simulation

[1]The amount of memory is $\Omega(\sum_{h=1}^{d} D \times G \times n \times K \times T_h)$, because of the allocation of the multiple simulated dynamics.

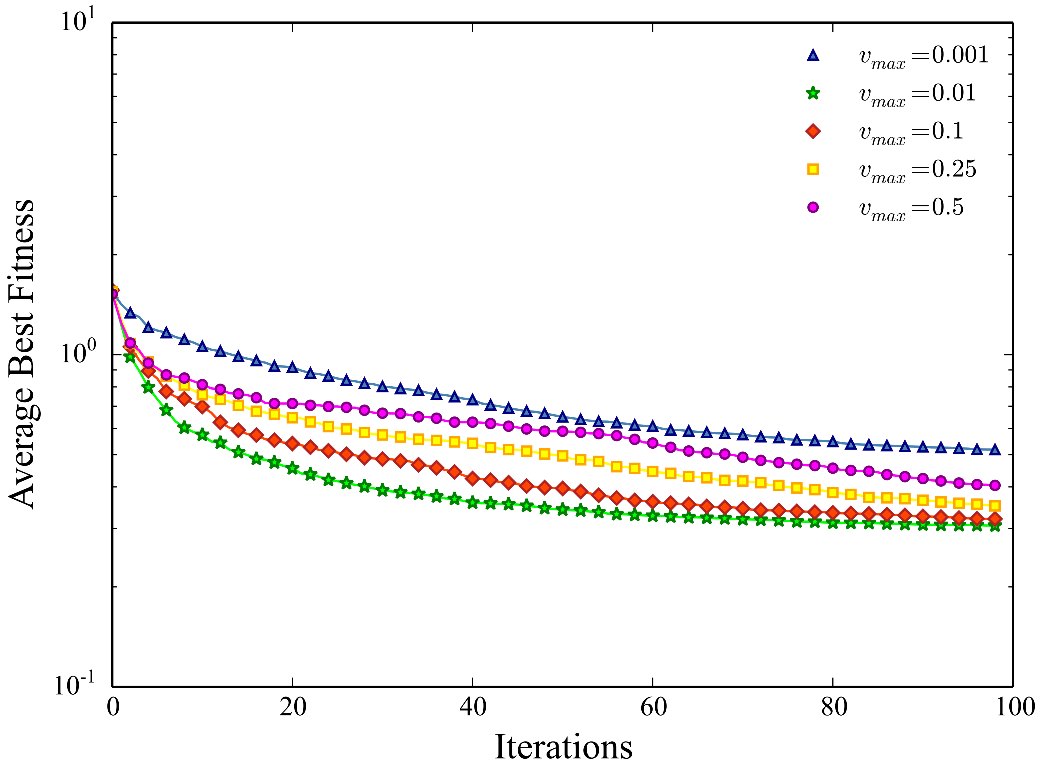


Figure 6.11: ABF of the best particles in the multi-swarm PSO, using different values of the maximum velocity v_{max} to estimate the stochastic constants of the PGN model. Differently from the MM model, these results show that a lower value of the maximum velocity, $v_{max} = 1/100$, allows both fast convergence and good estimation.

on the CPU. However, since the SMXs are finite resources, an excessive value G may saturate the GPU and cause a reduction of computational performances, which is not counterbalanced by any improvement in the optimization quality.

For instance, considering the case of the MM model, the execution of a single iteration using the best cuPEPSO settings requires $64 \times 30 \times 4 = 7680$ simulations to calculate the fitness values. Using the stochastic constants given in Table A.1 and the initial conditions of swarm σ_1, this batch of simulation takes 0.014 sec on a GPU Nvidia GeForce GTX 590 equipped with 1024 cores, while it takes 3.68 sec on a CPU Intel Core i7-2600 with a clock rate of 3.4 GHz. This corresponds to a 250× speedup, which represents a huge increment in performances, even with a desktop consumer video card. Using high-end machines (e.g., Nvidia Tesla GPUs), the speedup can be further increased. As the technology improves, larger numbers of parallel simulations can be instantiated without any impact on the overall running time.

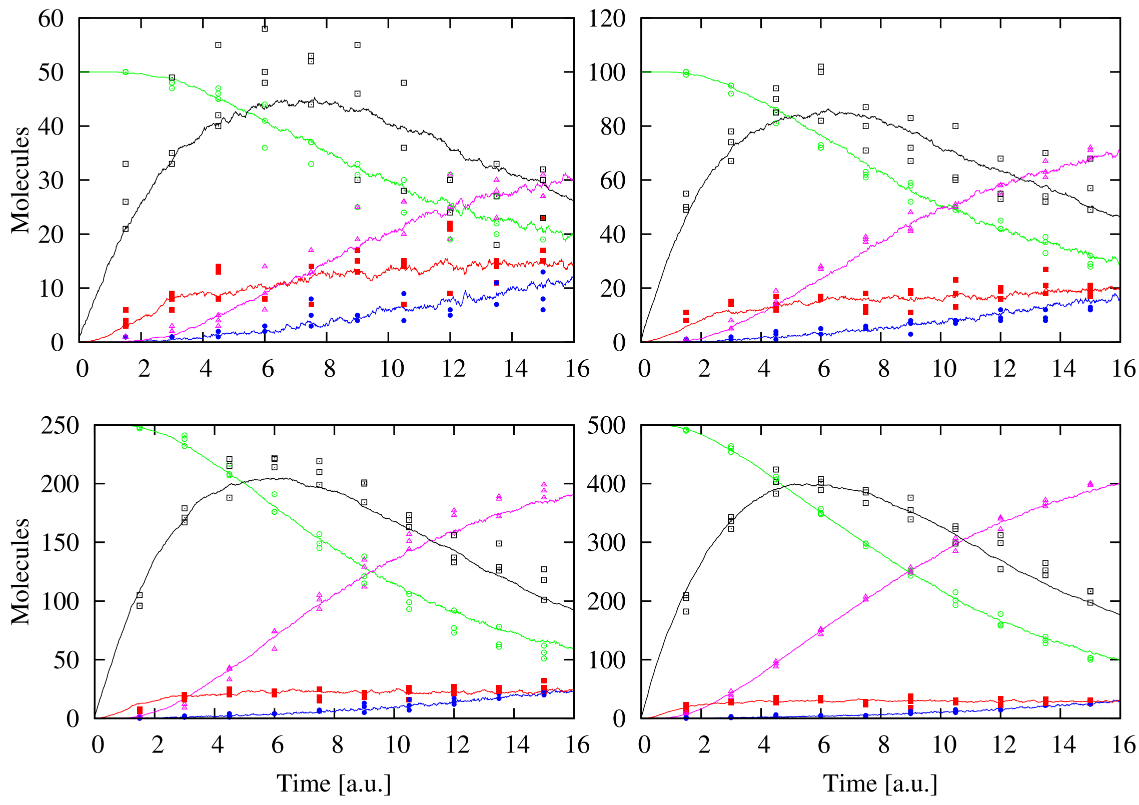


Figure 6.12: Comparison of DTTS (dots) with the simulated dynamics (solid lines) of species DNA (green), P_2 (blue), $DNA{:}P_2$ (pink), $mRNA$ (black) and P (red) of the PGN model. The dynamics were generated with tau-leaping using the stochastic constants of the multi-swarm best particle, and considering the different initial conditions characterizing each swarm. From top to bottom, left to right, the results in swarm σ_1, σ_2, σ_3, σ_4 are shown.

6.3 Discussion

In this chapter it was proposed an efficient method for the estimation of reaction constants in stochastic models of cellular systems, accelerated both algorithmically and architecturally by means of tau-leaping algorithm and GPGPU computing. cuPEPSO can handle experimental DTTS measured in different initial conditions, not requiring a uniform time sampling of target species, nor the observation of every biochemical species occurring in the system. The result of the estimation process carried out by cuPEPSO is a single parameters vector, that is able to simultaneously fit all measurements in all conditions. cuPEPSO is based on a multi-swarm version of PSO, where the populations can converge to a common solution thanks to a periodical exchange of their best particles.

The best solutions for the PE problem are identified by considering the lowest value of the fitness function. Other PE methods rely instead on the evaluation of the error between the inferred model parameterization and the expected (reference) set of parameters [349]. Anyway, this strategy implies that a reference parameterization is always known, or already available, which does not usually happens for real biological systems. In addition, any reference parameterization cannot always be considered as the most plausible solution, since there can be different model parameterizations — corresponding to different biological states of the systems — that can equally fit the target experimental data.

The estimation performances of cuPEPSO depend on the choice of its settings, for instance the number of particles in the swarms and the interval of migration. Since a good choice for these and other factors is crucial for the effectiveness of the methodology, a series of tests was performed to identify the optimal settings for cuPEPSO. A PE performed with the optimal settings yielded better results — in terms of fitting with the DTTS — than the one performed with the default settings, proving the usefulness of the meta-optimization process. Results showed that a subset of the settings cannot be considered effective as a rule, but should be fine tuned for different biological models. More precisely, the performances of cuPEPSO improve when the maximum velocity and the number of simulations used to determine the average dynamics are chosen according to the characteristics of the fitness landscape. Arguably, this is due to the variance of the stochastic dynamics, which is then reflected in the ruggedness of the fitness landscape. A procedure for the automatic choice of these settings, based on a global SA of the fitness function in the search space, is currently under investigation [49].

The problem of choosing the optimal settings for cuPEPSO, and in particular for PSO [308], might be solved by substituting the PSO with some parameters-free optimization algorithm. A variant of PSO, named TRIBES [61], avoids the manual tuning of its settings, by letting particles adapt their inner functioning strategy according to their performances. Since the performances of TRIBES were proven competitive with traditional PSO [78], an integration of this algorithm in cuPEPSO could be implemented, in order to achieve a black-box parameter estimator not requiring any manual setting.

With respect to the PE method previously presented in [26], cuPEPSO considers a more realistic experimental target to perform the optimization, as well as a simplified definition of the fitness function which yields better estimation results. This work also represents a major improvement of the PE method introduced in [232], and it relies on some interesting refinements of the whole method: cuPEPSO exploits the efficiency of tau-leaping instead of SSA; it considers a “normalized” fitness function, whereby all measured molecular species equally contribute to the estimation process; it averages the outcomes of the stochastic simulations, in order to reduce the noise of the dynamics.

To tackle its relevant computational complexity, cuPEPSO was conceived around the GPGPU computing architecture, to exploit the intrinsic parallelism of PSO on a single GPU-equipped workstation, instead of a typical distributed and parallel architectures. By spawning multiple threads per particle, the method achieves a significant boost with respect to a strictly sequential implementation. In particular, tau-leaping itself allows a relevant increment of performances, since it is usually faster than SSA. Being responsible for the largest part of the computational effort of cuPEPSO, the optimized GPU-oriented version of tau-leaping, cuTauLeaping, was exploited. An empirical comparison — carried out by running 512 parallel simulations of the MM model — evidenced that the GPGPU implementation of multiple parallel tau-leaping simulations is about 4 times faster than the SSA-based method introduced in [232].

In spite of the speedup achievable with tau-leaping, stochastic simulation is computationally demanding, since it requires the generation of an amount of Poisson-distributed random deviates proportional to the number of reactions of the model, that are needed to determine the reactions firing in each time step [46]. This problem can be mitigated by using normal distributions instead of Poisson distributions (e.g., the CLE) [103]. Anyway, this approximation is feasible for systems characterized by high concentrations of the molecular species. Nevertheless, it is worth noting that cuPEPSO is completely independent from the stochastic simulation methodology, thanks to a strict decoupling from the multi-swarm PSO, so that any GPU-powered biochemical simulator can be easily embedded into the methodology. For instance, the simulation of the dynamics of reaction-based models could be carried out using cupSODA [230] (see Section 5.1).

cuPEPSO is currently being extended by considering novel definitions for the fitness function, in order to properly take into account complex emergent behaviors of biological systems, such as oscillations or bistability. For instance, new fitness functions based on frequency-domain analysis or on the distance between empirical and target distributions are under development (e.g., by means of the Kolmogorov-Smirnov statistic [202]). Anyway, these fitness functions would require the generation of a huge set of experimental data, which is currently unlikely due to technical limitations of laboratory methods.

The multi-swarm, GPU-based method implemented in cuPEPSO indeed represents a novel methodology in the context of PE, and a useful tool for the computational analysis of biological systems. In this chapter the performance and effectiveness of this method were tested on two stochastic models of basic cellular processes, as a first step towards its application to larger biological systems, consisting of many reactions and many species, which are already under investigations in our research group [11, 27, 255].

Chapter 7

Reverse Engineering of biochemical systems

In the field of Systems Biology, the reverse engineering (RE) problem consists in the identification of the network of interactions among the basic components of a biological system. A number of computational methods for the automatic solution of RE have been developed to date, mainly oriented to the reconstruction of gene regulatory networks, which exploit experimental measurements obtained using genome-wide high-throughput techniques [18, 86].

If "simple" interaction networks are to be inferred (see Chapter 2), not requiring the estimation of kinetic parameters, then correlation-based RE methods [43], relying only on steady-state data, can be successfully applied. In such methodologies, some perturbations of the system (e.g., variation of metabolites concentration) can be used to calculate correlation coefficients and entropy-based mutual information values, which are exploited to build a putative interaction network (i.e., two metabolites are connected by an edge in the graph describing the network if their correlation is high). Then, the network is simplified and pruned by identifying and removing indirect interactions.

Arkin *et al.* proposed a modified approach of correlation-based RE relying on time-series data, which are used to calculate time-lagged correlations between metabolites [16]; the final network topology is obtained by applying an arbitrary threshold which removes the edges with lower correlation. Time-lagged correlation was exploited also in [66], coupled with probabilistic model calibration automatically identify and remove false positive edges. The latter are determined by calculating those reactions which have an unlikely, or null, kinetic constant. They also provide examples of more sophisticated alternative approaches relying on transfer entropy [303], calculated between couples of time-series of the chemical species.

In the specific context of metabolic systems, a way to detect indirect interactions and to perform the RE of a pathway whose edges in the interaction network have a specific *directionality* consists in analyzing the concentration profile of metabolites [340]. This method works by increasing the concentration of a metabolite and analyzing its impact on the time-series of the rest of metabolites.

These large interaction networks of metabolic pathways or gene regulation, anyway, are not suitable to fully characterize and comprehend the actual mechanisms that govern the functioning of biological systems. To this purpose, mechanistic models (Section 2.2) should be preferred, since they are based on the identification of the actual biochemical reactions that describe the physical interactions among all the chemical species occurring in the system (genes, proteins, metabolites, etc.). In contrast to large but simple interaction networks, these models require a proper parameterization to the aim of simulating the dynamics of the system and analyzing its behavior under different conditions [9, 355].

Mechanistic models are usually defined thanks to human expertise, by relying on pre-existing knowledge and available experimental data of molecular interactions. However, most of the times there is a general lack of knowledge concerning the exact molecular mechanisms occurring in living cells, therefore impelling the development of automatic RE methodologies, in order to devise a plausible network of biochemical reactions able to reproduce the experimental observations. In addition, even when adequate information on the network structure is known, the kinetic parameters required to reliably derive the system dynamics are usually either unavailable or uncertain, a fact that leads to the problem of PE [59], discussed in Chapter 6.

In this context, a novel methodology to automatically reconstruct the parameterized kinetic reaction network of a target biological system is presented in this chapter. The method, named cuRE, takes advantage of a small set of time-course measurements of the chemical species as the only target information and exploits the integration of two EC methods: CGP [214] to solve the RE problem, and PSO [157] to solve the PE problem (see Section 3.2 for a detailed explanation of these optimization techniques).

Solving RE and PE by means of two EC methods can favor the preservation of the best structures found throughout the evolutionary process: PSO can identify the best parameterization *for each* candidate network, so that a potentially good topology found by CGP is not discarded, a circumstance that could instead arise in the case of parameters co-evolution realized by the CGP itself. This "two-level separation" used for the reconstruction of parameterized reaction networks was previously shown to improve the final fitness and to prevent the premature structural convergence [178].

Up to now, EC has been successfully applied to tackle the PE problem [216]; for instance, Dräger *et al.* [78] have recently shown that the most efficient algorithm for PE is PSO [157], as also previously supported in [26] in comparison with GA, and exploited in [232, 238] with the multi-swarm topology described in Chapter 6.

Concerning the RE problem, also GP (see Section 3.2.3) has been exploited for the inference of biochemical reaction networks [13, 166, 178, 197, 285, 326] and of S-systems [57], which are generally modeled as systems of ODEs [143]. GP is a modification of GA in which individuals are programs, instead of being specific solutions to a problem [168]; these programs are generally described by recursive data structures, e.g., derivation trees or LISP s-expressions. The rationale is that such kind of data structures allow an intuitive definition of crossover operators as the exchange of two random subtrees; moreover, this representation can be directly interpreted without the need of a further decoding step. GP allows a higher level of problem solving, since programs can contain variables and, as such, they represent solutions to whole classes of problems. Nonetheless, one limitation of traditional GP is that it was not designed to deal with programs that are naturally represented by graphs.

To overcome this limit, CGP exploits individuals encoded as fixed-length vectors of integer numbers, which are mapped to directed graphs rather than derivation trees [214]. In this chapter CGP is exploited instead of GP for the RE problem since a graph is conceptually close to a network, and this mapping also allows a direct decoding of the solution into a human-comprehensible set of reactions. Moreover, the fixed length of the individuals of CGP does not need any explicit mechanism to contain the bloat, that is, the pathological and uncontrolled growth of the size of individuals in the population, that usually occurs in GP. It is important to point out that, even though the individuals of CGP have a fixed length, the graphs that they represent can have different size.

The main contents of this chapter are published in conference proceedings [228, 229, 234].

7.1 RE by means of CGP and PSO

In this section the cuRE methodology is fully described, considering the integration of CGP (Section 3.2.4) for the reconstruction of the network and PSO for the estimation of its parameters (see Sections 3.3.2 and 6.1.3).

CGP is used here to reconstruct the network of biochemical reactions using as target a set of experimental measurements, whose temporal evolution is fitted by executing

PSO to solve the associated PE problem — that is, PSO automatically determines the kinetic parameters of the network identified by CGP. Since, in general, routine laboratory experiments sparsely sample the chemical species and only a few repetitions of the experiment are carried out (see, also, Section 6.1.1), cuRE was developed to match these specifications. To the best of my knowledge, this is the first time that CGP is exploited in the context of the RE of biochemical networks, and that CGP and PSO are successfully integrated to solve the RE and PE problems simultaneously; in particular, these two evolutionary methods are applied *from scratch*, that is, no previous information on the system is considered, besides the measured concentrations of its chemicals.

In what follows, the notation for CGP, introduced in 3.2.4, is used. To solve the RE problem, CGP was implemented using $n_n = 2$ functional nodes and setting $\mathfrak{F} = \{+, -\}$. The set of input nodes is determined by the augmented set of chemical species $\hat{\mathcal{S}} = \mathcal{S} \cup \{\emptyset\}$ (i.e., $n_i = |\hat{\mathcal{S}}|$), where $\emptyset$ denotes the null species used to model sink and source reactions (see also Section 2.2.1). By composing input and functional nodes, CGP encodes complex expressions that can be converted, by means of symbolic calculation, into simple arithmetical equations, as described below.

In CGP, the number of output nodes n_o is fixed. Since the actual number of reactions of the target system is unknown, a proper choice for n_o is fundamental because it represents the upper bound to the number of reactions that a candidate network can contain. As a heuristic, n_o is set equal to the number of all possible bimolecular reactions that can occur in the system (also considering the empty species), i.e., $n_o = \binom{|\hat{\mathcal{S}}|}{2}$. Furthermore, in the work presented in this chapter it is assumed $l = n_c$, that is, the output nodes can be connected to any functional or input node, and each functional node belonging to column j can be connected to any other node between column $j-1$ and the input nodes.

The pseudocode of cuRE methodology based on CGP is reported in Figure 7.1. The process starts with the creation of a population of $I = 1 + \lambda$ random CPs (lines 1–3). Then, the population undergoes the ES process (lines 4–29) which can be decomposed in three main steps:

- in **Step 1** each CP is converted into a network of chemical reactions, then the PSO estimates its parameters and calculates the fitness value of the parameterized candidate network (lines 5–8);
- in **Step 2** the CPs are ranked according to their fitness values, in order to identify the best CP (line 9);

```
for 1 to I do
   CGP_population.add(create_CP($\hat{\mathcal{S}}, n_o, n_r, n_c$))
end for
for CGP_generation ← 1 to 100 do
   for all CP in CGP_population do
      network ← CP.convertToNetwork()
      CP.fitness, CP.parameters ← PSO(network)
   end for
   best_CP ← find_best_individual(CGP_population)
   CGP_population ← []
   CGP_population.add(best_CP)
   for 1 to I-1 do
      repeat
         repeat
            candidate_CP ← best_CP.mutation($\rho$)
         until candidate_CP.connected()
      until candidate_CP not in CGP_population
      for all reaction in candidate_CP do
         if not consistent(reaction) then
            candidate_CP.remove(reaction)
         end if
      end for
      CGP_population.add(candidate_CP)
   end for
   if fitness(elite_CP) < fitness(best_CP) then
      elite_CP ← best_CP
   end if
   CGP_population.add(elite_CP)
end for
return bestCP
```

Figure 7.1: Pseudocode of cuRE.

- in **Step 3** a new population is formed, whose individuals are the best CP (lines 10–11) and the $I-1$ offspring created by randomly mutating the best CP (lines 12–24).

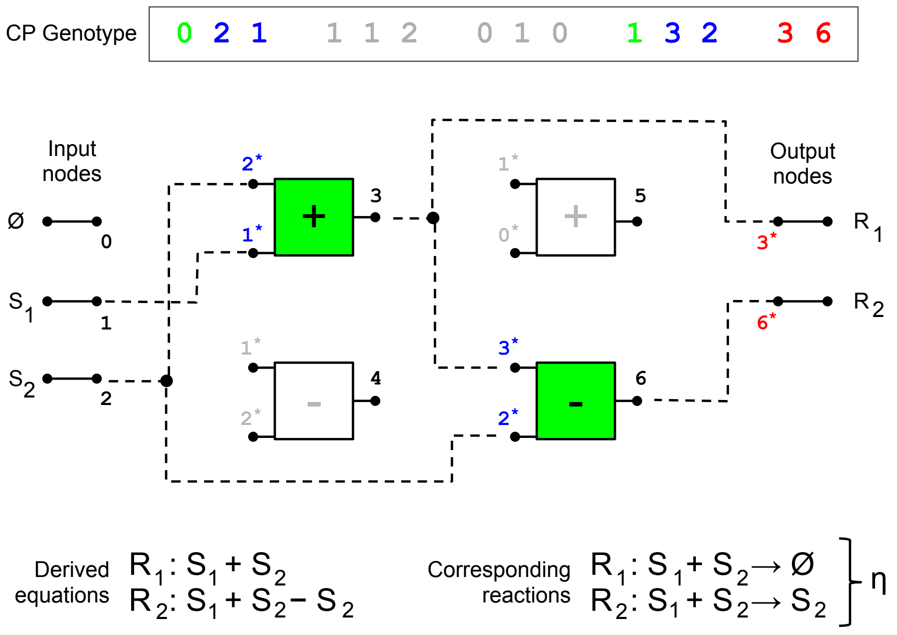
The mechanism for the conversion of a CP into a RBM (**Step 1**) is schematized in Example 7.1.1. First, the connectivity of the nodes is determined: for each output node, its connections are followed backwards by recursively passing through the functional nodes, constructing a derivation tree. Then, the derivation tree is translated into an arithmetical equation in which the positive terms (respectively, negative) are considered as reactants (products), yielding a candidate chemical reaction for the network. In this process, it is assumed that the null species $\emptyset$ does not contribute to the derived reaction, that is, $S_s \pm \emptyset = S_s$ and $\emptyset - S_s = -S_s$, for any $S_s \in \mathcal{S}$. By repeating the process for all output nodes, a candidate RBM η is obtained. This particular encoding — in which the functional nodes operate only on input nodes corresponding to chemical species or some other functional nodes — ensures that the type consistency property (Section 3.2.3) is always satisfied. Moreover, since this representation allows to encode any type of reaction (according to MAK), the set $\mathfrak{F}$ also ensures the sufficiency property.

In the work presented in this chapter, the fitness function is based on the comparison of the target series against a simulated dynamics, which is generated *in silico* according to the candidate network, by exploiting the LSODA algorithm. Since the reaction network derived with CGP is not complete until a proper kinetic parameterization is also given, the fitness evaluation of each candidate solution embeds a PE phase, which is performed by means of PSO. Let $X_s(t_c)$ be the experimental concentration at time t_c of the chemical species $s \in \mathcal{S}$, and let $Y_s^{\boldsymbol{\gamma}}(t_c)$ be its simulated concentration, obtained with cupSODA using a putative parameterization $\boldsymbol{\gamma}$ for the model η, sampled at time t_c. The fitness function of PSO for a particle $\boldsymbol{\gamma}$ is based on the least squares:

$$\mathcal{F}(\boldsymbol{\gamma}) = \sum_{c=1}^{C} \sum_{s=1}^{|\mathcal{S}|} \left(Y_s^{\boldsymbol{\gamma}}(t_c) - X_s(t_c)\right)^2, \tag{7.1}$$

where $C \in \mathbb{N}$ is the total number of samples in the target time-series. The fitness values of CGP's candidate solutions are calculated at the end of the PE process, performed by means of PSO: when the PSO termination criterion is met — in this case, after 300 iterations — the value of $\mathcal{F}(\mathbf{g})$, that is, the fitness of the best solution $\mathbf{g} = (k_1, \ldots, k_M)$ determined by PSO for the PE problem, is taken as the fitness of the network η. In other words, it represents the fitness of the candidate solution of the RE problem solved by CGP. Therefore, with abuse of notation, it is set $\mathcal{F}(\eta) := \mathcal{F}(\mathbf{g})$. So doing, the ranking of the CPs is used to identify the best solution (**Step 2**).

Example 7.1.1 *Example of the conversion of a CP into the final RBM. The connectivity of each node in the graph relies on the genotype of the CP. Functions + and − in the grid are represented by 0 and 1 in the genotype, respectively (first number of the four triplets). The genotype is generated by randomly drawing values related to the chemical species (0,1,2) and to the output of each functional node (3, . . . , 6), and then assigning these values to the input of the functional nodes and to the output nodes (numerical values with asterisk). Note that, in the construction of the genotype, given a functional node only the values of previous nodes (both input and functional) can be randomly selected and assigned to its input connections. The graph is parsed backwards, from the $n_o = 2$ output nodes R_1, R_2 to the $n_i = 3$ input nodes $\emptyset, S_1, S_2$. The resulting equations are converted to an equivalent set of biochemical reactions, whose kinetic constants will be estimated by PSO. The green functional nodes are involved in the equations, because a path from the input nodes to the output nodes exists here; the white functional nodes, instead, are not connected and do not participate to any equation in this CP. Thus, their corresponding genes (gray numbers) are non-coding sequences.*



The generation of new CPs is accomplished by using the best solution and applying a mutation operator. Once a new individual η is created, cuRE verifies whether the network of reactions is constituted by a single connected component or not (line 16); stated otherwise, η has to contain at least one reaction whose reactants or products do not participate in any other reaction of η. If a network contains more than a single connected component, then the dynamics of a subset of its chemical species turns out to be independent from the other species. To avoid this, a CP encoding a non-connected network is reverted and mutated again, until this condition is satisfied. The network η is then compared to the rest of the population: if it is identical to some other CP, it is mutated again in order to maintain diversity, and the process is repeated (line 17).

When a new population is generated, the consistency of the reactions of each candidate solution η are checked (lines 19-22). Thus, each reaction R appearing in η must obey the following conditions:

- it is not allowed that $a_i = 0$ and $b_i = 0$ for all $i = 1, \ldots, N$ (that is, reactions of the form $\emptyset \rightarrow \emptyset$ are invalid);
- reactions of the form $R_\mu : a_i S_i \rightarrow b_i S_i$, for any a_i and b_i are excluded, since they correspond to unfeasible biochemical processes where a_i molecules of species S_i are converted into b_i molecules of the same species;
- at most second-order reactions (that is, reactions that have no more than two reactant molecules) are allowed, since third-order reactions have a probability to occur almost equal to zero, as they would require the simultaneous collision of three reactant molecules[1].

These conditions, together with the condition that no identical reactions appear in η, allow to strongly reduce the search space. In particular, the condition related to the maximum order of reactions allowed in the network η permits to control the maximum length of the CP expressions, which is realized by setting $n_c \leq 3$. In general, if a reaction R is not consistent, it is removed from the network, so that $\eta = \eta \setminus \{R\}$. This robust decoding process ensures that the evaluation safety property, described in Section 3.2.3, is always satisfied.

Finally, when a new consistent CP is produced, it is added to the new population (line 23). At the end of **Step 3**, to the aim of increasing the convergence speed, elitist selection is applied, that is, a non-mutated copy of the best candidate network found so far is added as the $(I + 1)$-th individual of the population (lines 25–28). This iterative process is repeated for 100 generations and, at the end, the CP with the best fitness, along with its parameterization, is the result of the RE problem.

For what concerns the PE, the CGP presented here embeds the PSO with the following settings: 30 particles, $C_{soc} = C_{cog} = 2.05$, as suggested in [78]; inertia linearly

[1]This choice does not pose a limitation to the practical applicability of cuRE, since any third-order reaction can be decomposed into a cascade of consecutive reactions of lower order.

decremented from $w = 0.9$ to $w = 0.4$; search space bounded between $1 \cdot 10^{-10}$ and $1 \cdot 10^{1}$ — for each kinetic parameter — with damping boundary conditions, as suggested in [361]. The settings for the upper bound of the search space for the kinetic constants of chemical reactions is fundamental for the entire optimization process, since a limited range of variation might exclude the global optimum of the problem under investigation. However, in the case of chemical systems composed of a small set of reactions that involve low molecular concentrations (as in the case of the systems listed in Table 7.1), limiting the search space of kinetic constants could facilitate the optimization process. This is motivated by the fact that high values of kinetic constants can lead to meaningless dynamics where the species are entirely consumed within the first instants of the simulation; nevertheless, it is important to note that narrowing the search space might exclude the optimal solution.

7.1.1 Results

cuRE was applied for the RE of a set of RBMs, taken from [68]. These network models, listed in Table 7.1, are representative of reactions modules that usually occur in biochemical systems: the conversion of a chemical species into another species (N_1), a cascade of conversions (N_2), a feedback mechanism (N_3), the dissociation of a complex (N_4), two different branched pathways (N_5, N_6), and a closed branch (N_7). Network N_8 was instead taken from [166], it represents a real chemical system consisting of three reactions involved in the synthesis and degradation of ketone bodies [282][2].

Table 7.1 reports the set of reactions involved in each network and the initial concentration of the chemical species. For what concerns the kinetic constants, all values were arbitrarily set to 1 (being a different choice of kinetic constants not able to impair the PE process for such small networks), except for N_8 where reactions constants have the following values: $k_1 = 1.56, k_2 = 0.85, k_3 = 0.7$ (according to [166]).

The CGP settings used during the application of cuRE are: $n_r = 5, n_c = 2$, $I = 10$ individuals, 100 generations, mutation rate $\rho = 0.20$. The target series were produced by sampling 20 points from ODE simulations of length $t = 5$ [a.u.].

cuRE correctly inferred the structure of networks $N_1, \ldots, N_7$, showing as well a good approximation of their reference parameterizations. For each network, the values of the estimated kinetic constants are: N_1: $k_1 = 0.951$; N_2: $k_1 = 0.949, k_2 = 0.954$; N_3: $k_1 = 0.968, k_2 = 0.973, k_3 = 0.969$; N_4: $k_1 = 0.998$; N_5: $k_1 = 0.986, k_2 = 0.964$; N_6: $k_1 = 1, k_2 = 1.001, k_3 = 1$; N_7: $k_1 = 1.001, k_2 = 0.999, k_3 = 1.008$. The quality of

[2]The chemical species $S_1, \ldots, S_4$ correspond to Acetoacetyl-CoA, Acetyl-CoA, an intermediate complex (INT-1) and Acetoacetate, respectively.

Table 7.1: Biochemical reaction networks tested for RE.

Network	*List of reactions*	*Initial state*
N_1: Conversion	$S_1 \rightarrow S_2$	$[S_1] = 1 \cdot 10^{-4}$, $[S_2] = 5 \cdot 10^{-5}$
N_2: Cascade	$S_1 \rightarrow S_2 \rightarrow S_3$	$[S_1] = 1 \cdot 10^{-4}$, $[S_2]$=0, $[S_3]$=0
N_3: Feedback	$S_1 \rightarrow S_2 \rightarrow S_3, S_3 \rightarrow S_1$	$[S_1] = 1 \cdot 10^{-4}$, $[S_2]$=0, $[S_3]$=0
N_4: Dissociation	$S_1 \rightarrow S_2 + S_3$	$[S_1] = 1 \cdot 10^{-4}$, $[S_2]$=0, $[S_3]$=0
N_5: Branch 1	$S_1 \rightarrow S_2, S_1 \rightarrow S_3$	$[S_1] = 1 \cdot 10^{-4}$, $[S_2]$=0, $[S_3]$=0
N_6: Branch 2	$S_1 \rightleftarrows S_2, S_1 \rightarrow S_3$	$[S_1] = 1 \cdot 10^{-4}$, $[S_2]$=0, $[S_3]$=0
N_7: Closed branch	$S_1 \rightarrow S_2 \rightarrow S_3, S_1 \rightarrow S_3$	$[S_1] = 1 \cdot 10^{-4}$, $[S_2]$=0, $[S_3]$=0
N_8: Ketogenesis	$S_1 \rightarrow S_2, S_1 + S_3 \rightarrow S_4$, $S_4 \rightarrow S_2 + S_3$	$[S_1] = 1 \cdot 10^{-4}$, $[S_2]$=0, $[S_3] = 1 \cdot 10^{-5}$, $[S_4] = 1 \cdot 10^{-5}$

these estimated parameterizations, and of the RE problem in general, was assessed for networks $N_1, \ldots, N_7$ by comparing the simulated dynamics of each network — carried out using LSODA algorithm with these values — with the target time-series, generated using the reference parameterizations previously given.

Concerning network N_8, Figure 7.2 shows the comparison of the target network (left side) and the one produced by cuRE (right side). The dashed arrows indicate the interactions between chemical species that were not properly reconstructed. Notwithstanding the difference in the network structure, in Figure 7.3 it is shown that the simulated dynamics of the reconstructed network perfectly fit the target time-series of all four chemical species. This is due to the fact that the lack of the arc colored in red and the presence of the arc colored in green in the right (reconstructed) network, was compensated during the RE process by the addition of the reaction colored in blue. Moreover, the dynamics of the two networks perfectly match also thanks to the precise estimation of the kinetic constants of reactions R_1 and R_2.

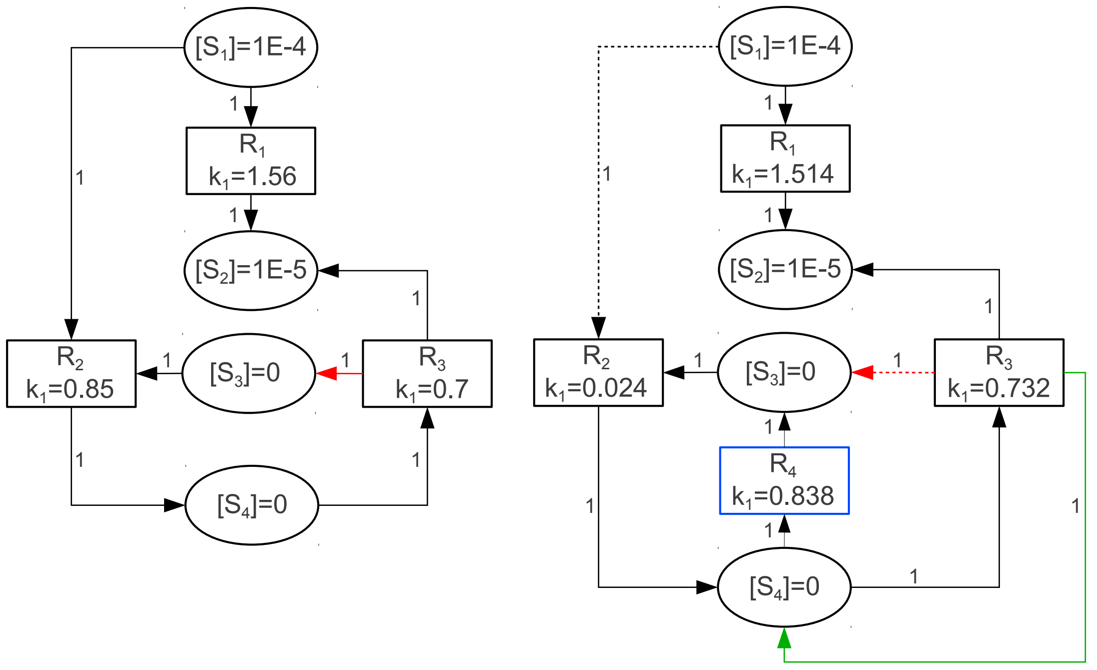


Figure 7.2: Comparison of the target network N_8 (*left*) and the RBM produced by cuRE (*right*). The dashed arrows indicate the interactions between chemical species that are not correctly reconstructed.

To understand the impact on the RE performance of the setting of n_r and n_c, that is, the dimensions of the CPG grid, a series of tests were executed. Figure 7.4 shows the convergence of cuRE according to different values of n_r and n_c, whose best choice ultimately depends on the dimensions of the network to be inferred. It is worth noting that a too small grid (e.g., 5×2) can impair the RE performance, since the CGP has less freedom of movement inside the search space. A larger grid (e.g., 8×3) does not help the convergence either, since it is more likely to generate long equations which could be discarded from η as non consistent, that is, equations corresponding to reactions that do not satisfy the constraints described in Section 7.1.

Figure 7.4 is interesting because it shows another relevant characteristic of cuRE, connected to the fitness evaluation. The fitness value for a candidate network η is calculated as the fitness of a PSO optimization. Being the latter a stochastic algorithm, two different PE executions on the same network η have a high probability of yielding two different parameterizations which, in turn, are likely to have two different fitness values: this is the explanation of the large fluctuations in Figure 7.4 (top graphic).

To better understand the impact on the performances of different settings of the grid dimension, these fluctuations are reduced by calculating the best fitness value for

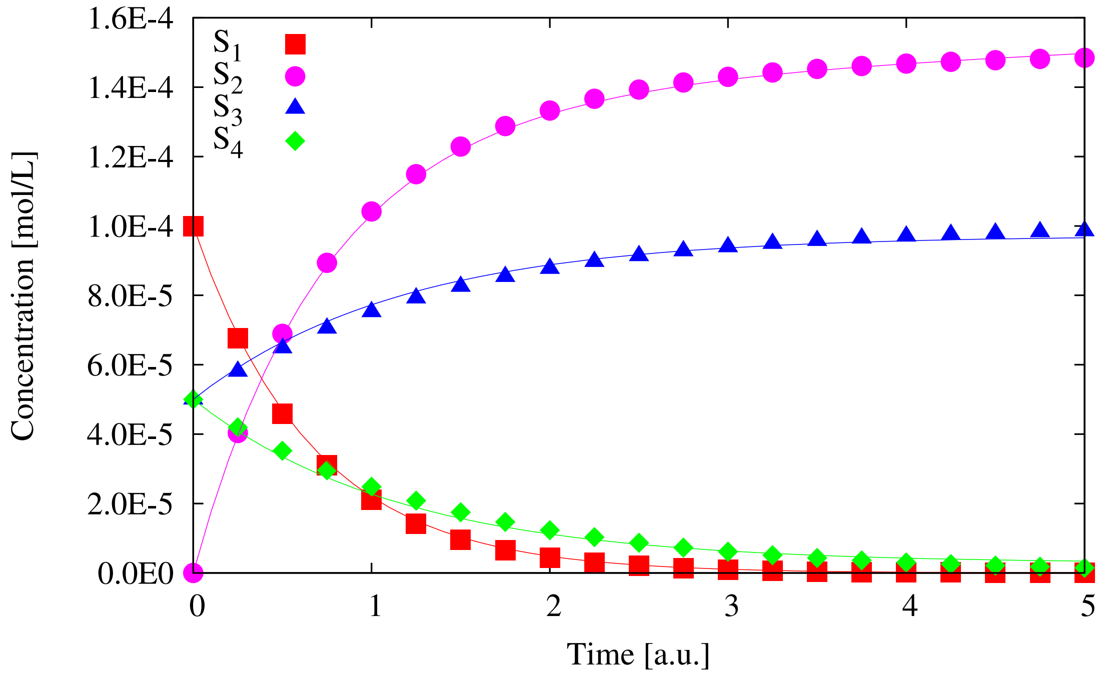


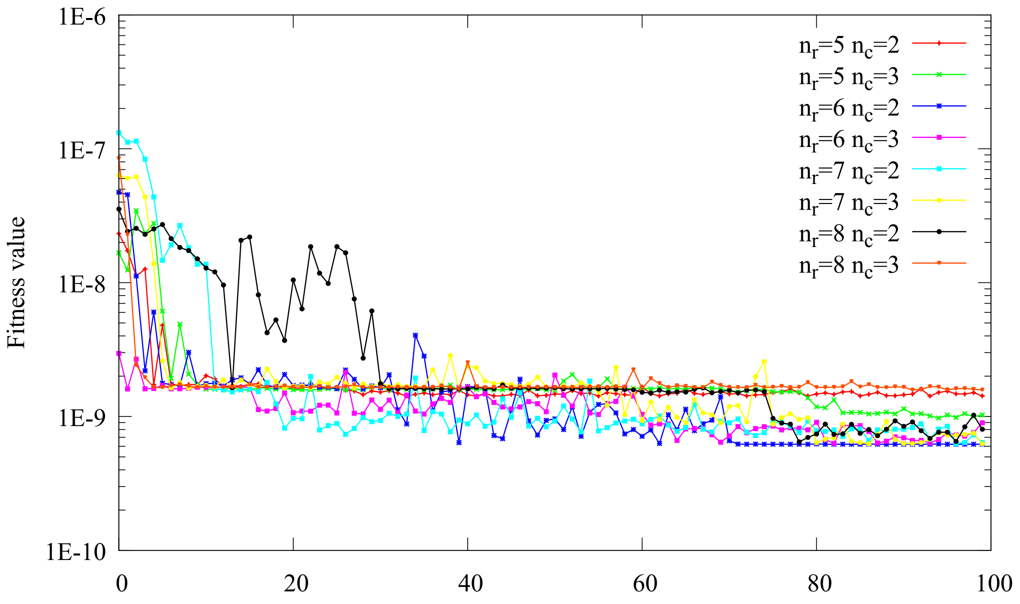
Figure 7.3: Comparison of the target time-courses of network N_8 (points) and the simulated dynamics of the RBN produced by cuRE (lines). Even though the two networks are different, their dynamics perfectly overlap.

the z-th generation ($z \in \mathbb{N}$) of CGP as:

$$\mathcal{F}(\eta_z)_{best} = \min_{z' \leq z} \mathcal{F}(\eta_{z'}), \tag{7.2}$$

where η_z denotes the best CP individual found during the z-th generation. Figure 7.4 (bottom graphic) represents the results according to Equation 7.2: it is now more evident how the choice of the grid dimension can negatively affect the convergence of the algorithm, as for cases (5×2) and (8×3). On the contrary, there exist grid configurations that seem to facilitate the RE and PE problems, as for cases (6×2) and (7×3). It is worth noting that the effectiveness of this result can be stated only for the kind of small networks considered in this section; a more general test should be performed on larger biochemical reaction systems, by increasing the size of the network while increasing as well the number of rows in the CGP grid. Stated otherwise, larger values of n_r in the CGP grid should be also investigated with biochemical networks having a higher number of reactions.

A further test was also carried out to further investigate the performances of cuRE, by considering the impact of the ρ setting, that is, the mutation rate. Results in Figure 7.5 show that the best choice is $\rho = 0.20$, which allows a good exploration of the search



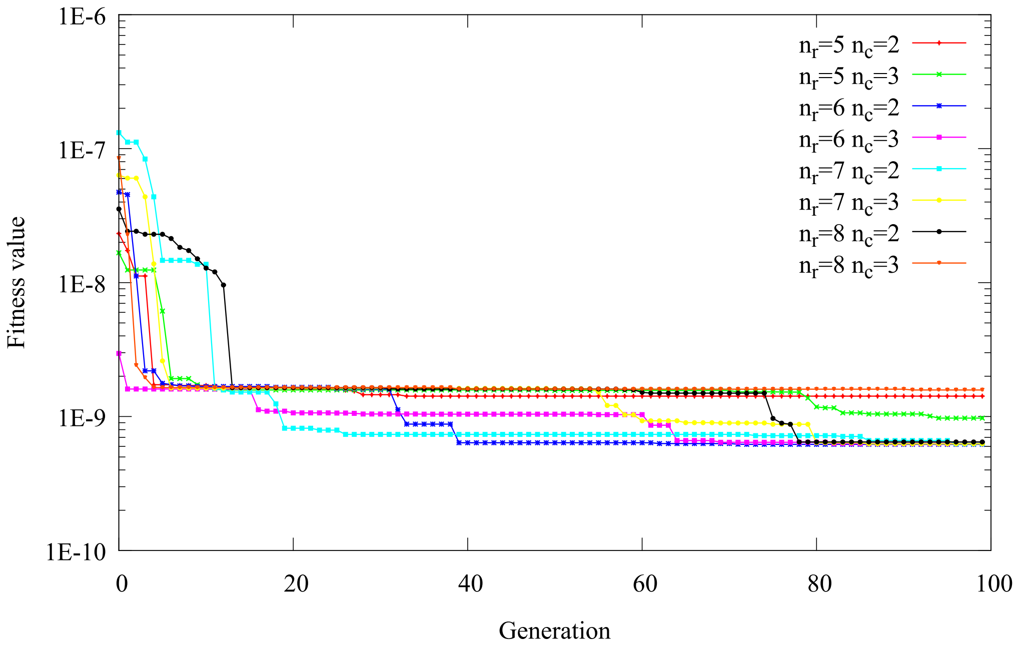


Figure 7.4: Fitness value of the best solution (top) and best fitness values (bottom), obtained by varying the settings for n_c and n_r (i.e., the dimensions of the grid of functional nodes). The fitness value on the y-axis is on a logarithmic scale. The top graphic highlights the presence of fluctuations, which are the consequence of the stochasticity of PSO. The bottom graphic shows how a proper choice for the grid is fundamental for the convergence of the algorithm.

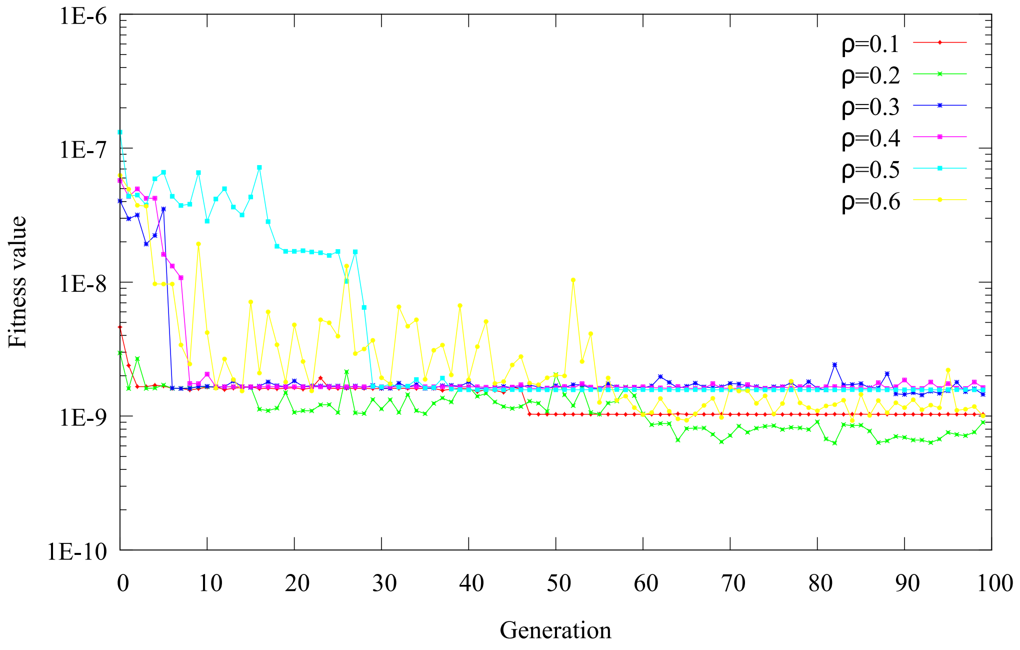


Figure 7.5: Fitness value of the best solutions found during the RE of network N_8, varying the settings for ρ (i.e., the proportion of a parent's genotype that is mutated during the offspring creation). The fitness value on the y-axis is on a logarithmic scale. This result shows that $\rho = 0.20$ is the best choice for this application.

space and achieves the best fitness values. This is in agreement with published results on CGP [214], where it is shown that the best performance is usually obtained by setting ρ in the range $0.15 - 0.25$. According to these results, low mutation rates seem to lead to a better convergence speed, even though, when the mutation rate is too small, the CGP cannot explore the search space properly: for instance, in the case of $\rho = 0.10$, the best individual stops improving at the 47-th generation out of 100. Vice versa, higher mutation rates allow to avoid the premature convergence, but can easily generate offspring with higher fitness values.

7.1.2 Discussion

From a computational point of view, cuRE methodology requires fewer fitness evaluations than most of the existing algorithms, because ES-based CGP can exploit a very small population. Being the fitness evaluation the most computationally intensive task, cuRE strongly reduces the required running time. Moreover, in order to further reduce the computational effort, the fitness evaluations (i.e., simulations and comparisons against target data) are performed on the GPU, by exploiting the ODEs solver cupSODA described in Section 5.1.

When some of the target chemical species involved in the reaction network have a very low molecular amount, in the order of a hundred molecules or less, a stochastic simulation methodology may be preferable to correctly reproduce the emergent phenomena due to stochasticity. It is worth noting that cuRE can be straightforwardly extended to infer biochemical systems characterized by stochasticity, by simply replacing the deterministic simulator with a stochastic simulator. As a matter of fact, since the RE method evolves RBMs, the simulation method is completely independent from the rest of the evolutionary process and can exploit a GPU-accelerated stochastic simulator like cuTauLeaping (Section 5.3).

Most of the times, to solve the RE problem no *a priori* information on the topology of the system is known; therefore, the optimization process relies only only on the dynamics of the chemical species. This information is sufficient to properly reconstruct the target time-series but, usually, it is not enough to discriminate between different network topologies [327]. As a matter of fact, the fitness of the reconstructed network is often comparable to the fitness of the target system. It is currently under investigation a methodology to convey new knowledge domain constraints, in order to better drive the RE process.

cuRE is also prone to phenomenological modification of the network topology, which represent local minima of the RE problem. This kind of situation must be better characterized, in order to implement some automatic correction algorithm. Some previous works tackled the problem of redundant reactions by embedding the Akaike Information Criterion into the fitness function [5], in order to exploit some parsimony considerations. Unfortunately, this strategy failed to achieve its goal, leading to a premature convergence to small networks which poorly fit to target data [178]. Thus, a different approach to the simplification of evolved networks needs to be proposed.

In general, the issue of the *indistinguishability* of equivalent networks is known [327], thus a good practice should be the multiple execution of the RE process, followed by an analysis of the obtained solutions to asses their biological plausibility. This *a posteriori* analysis could then help in the identification of a single network, or could lead to plan new biological experiments with the aim of better understanding the real functioning of the system. This issue will be extensively discussed in Section 10.3.

7.2 RE by means of Evolutionary Petri Nets

To the aim of developing an evolutionary methodology for the RE problem whose candidate solutions are based on PNs, it is hereby proposed an extension of the conventional PN formalism (Section 2.2.5), called *Evolutionary Petri Net* (EPN) [229]. EPNs provide a conceptual framework for the representation of RBMs, and embed robust and consistent genetic operators.

Before presenting the EPN formalism, some necessary elements have to introduced. A *Resizable Petri Net* (RPN) is defined as a 9-tuple $\xi = (P, P^h, T, T^h, F, W, M_0, O_{pre}, O_{post})$ where:

- $P = \{p_1, \dots, p_m\}$ is a finite set of places;
- P^h is the set of hidden places, such that $P \cap P^h = \emptyset$;
- $T = \{t_1, \dots, t_n\}$ is a finite set of transitions;
- T^h is the set of hidden transitions, such that $T \cap T^h = \emptyset$ and $(P \cup P^h) \cap (T \cup T^h) = \emptyset$;
- $F \subseteq \Big((P \cup P^h) \times (T \cup T^h)\Big) \cup \Big((T \cup T^h) \times (P \cup P^h)\Big)$ is the set of arcs;
- $W : F \to \mathbb{N}$ is a weight function, which associates a non-negative integer value to each arc;
- $M_0 : P \to \mathbb{N}$ is the initial marking of non-hidden places of the net (all hidden places have zero tokens, initially);
- $O_{pre} \in \mathbb{N}$ is the maximum pre-order allowed in the RPN, that is, for each $t \in (T \cup T^h)$
$$\sum_{\substack{p \in {}^\bullet t \\ p \in (P \cup P^h)}} W(p, t) \leq O_{pre}; \tag{7.3}$$
- $O_{post} \in \mathbb{N}$ is the maximum post-order allowed in the RPN, that is, for each $t \in (T \cup T^h)$
$$\sum_{\substack{p \in t^\bullet \\ p \in (P \cup P^h)}} W(t, p) \leq O_{post}. \tag{7.4}$$

Differently from a traditional PN, a RPN is composed of a fixed number of places ($|P| = m$) and transitions ($|T| = n$), together with a variable number of hidden places and transitions in the sets P^h and T^h, respectively, whose cardinalities can change during the evolutionary process because of the application of the genetic operators. Two examples of RPN are depicted in Figure 7.6a. Unlike other similar works on dynamically reconfigurable PNs [191], or on virtual PNs [218], RPNs are modified by “exogenous” mutations that can arbitrarily introduce new hidden places and new hidden transitions, where hidden transitions can represent events involving both elements from P and P^h. In RPNs the sets P and T are explicitly separated the from the hidden (modifiable) sets P^h and T^h in order to make use of the available, consolidated domain knowledge of the system under investigation in the form of static (i.e., non modifiable) places and transitions. In the case of zero-knowledge on the interaction of the elements of the net, the set P is populated with those elements that are known to be part of the system and whose data can be exploited by the fitness function, while $T = \emptyset$.

Conversely, hidden places and transitions are exploited by the genetic operators of EPN in order to explore alternative and more complex topologies of the net. Since hidden (i.e., modifiable) sets are dynamically modified and evaluated by the EC algorithm, EPNs allow the optimization of an existent PN according to some given constraints; moreover, they allow the automatic discovery of simplified or more efficient alternative models. These tasks can be accomplished by initializing the system of interest as a “fully-hidden” RPN (that is, $P = T = \emptyset$), and letting the evolutionary algorithm explore the space of alternative models.

The RPN formalism includes the pre- and post-order conditions (Equations 7.3 and 7.4) for the following reasons. First, they help to reduce the bloating phenomenon of GP by limiting the number of arcs. Secondly, they avoid the possibility of a convergence to degenerate or overfitting solutions, represented by completely connected PNs, by limiting the weights of in- and out-going arcs of transitions. This can also be used to limit the search space, by excluding *a priori* unfeasible topologies. However, both pre- and post-order conditions are optional and can be excluded from the RPN by setting $O_{pre} = O_{post} = \infty$.

Two classic genetic operators, crossover and mutation, are exploited by the evolutionary algorithm to modify RPNs. Given the space Ξ of all possible RPN topologies, an *Evolutionary Petri Net* (EPN) can be defined as a triple $E = (\xi, \chi, \mu)$ where:

- $\xi \in \Xi$;
- $\chi : (\Xi \times \Xi) \to (\Xi \times \Xi)$ is the crossover operator which modifies two RPNs ξ and $\bar{\xi}$, where ξ and $\bar{\xi}$ are such that $P_\xi = P_{\bar{\xi}}$;
- $\mu : \Xi \cup \{p_{in}, t, p_{out}\} \to \Xi$ is the mutation operator, where $\{p_{in}, t, p_{out}\}$ is a triple consisting of two places p_{in} and p_{out} and a transition t, namely $p_{in}, p_{out} \in$

$(P \cup P^h) \cup P^\infty$ and $t \in (T \cup T^h) \cup T^\infty$, where P^∞ and T^∞ are infinite sets of places and transitions, such that $P^\infty \cap (P \cup P^h) = \emptyset$ and $T^\infty \cap (T \cup T^h) = \emptyset$.

The functioning of χ and μ is described in the next sections, where the RPN at the g-th generation of the evolutionary process it is denoted by $\xi(\mathfrak{g})$, $\mathfrak{g} \in \mathbb{N}$.

7.2.1 Genetic operators

In this section, the two main genetic operators of EPNs — crossover and mutation — are described in detail. The selection operator, which is used to control the evolutionary process by deciding which RPNs will propagate their genetic material in the next generation, can be chosen among any of those described in Section 3.2.1 and, thus, it is not explicitly discussed in this chapter.

The crossover operator

The crossover mechanism implements the exchange of genetic material between two RPNs, in order to generate new offspring which inherit the best substructures of the parents. Various crossover mechanisms specifically designed for graphs were proposed in literature [172, 252], in particular to tackle the complex case of two networks with a different number of nodes [153] but, to the best of my knowledge, no specific work exists about the crossover between bipartite graphs or, more specifically, PNs. Indeed, the crossover between two PNs is supposed to identify some substructures in each graph, "detach" them from one parent graph and "attach" them into the other, and vice versa, while keeping the consistency of both graphs. The difficulties arise in:

- how to characterize a substructure;
- how to "detach" it from a graph;
- how to "attach" it to a new graph, considering that there is not a direct correspondence between the elements belonging to different sets of hidden places.

The last issue is extremely relevant, because it determines the ability of EPNs to transferring a precise functionality from a RPN to its offspring.

A possible crossover mechanism of two RPNs $\xi(\mathfrak{g}), \bar{\xi}(\mathfrak{g}) \in \Xi$, named *sticky crossover* (SC), could work on hidden transitions as follows: a transition $t_\chi \in T^h$ ($\bar{t}_\chi \in \bar{T}^h$, respectively) is randomly selected in the RPN ξ ($\bar{\xi}$) (the red nodes in Figure 7.6b), and the substructures consisting of the preset and postset of t_χ ($\bar{t}_\chi$) (dotted lines) are exchanged between ξ and $\bar{\xi}$. $P^+ = ({}^\bullet t_\chi \cup t_\chi^\bullet) \cap P^h$ ($\bar{P}^+$, respectively) denotes the set of hidden places contained in the substructure connected to t_χ ($\bar{t}_\chi$), that is added to $\bar{\xi}$ (ξ); $F^+ = \{(t_\chi, p), (p, t_\chi) \in F \mid p \in P^+\}$ ($\bar{F}^+$) denotes the set of arcs belonging to this substructure.

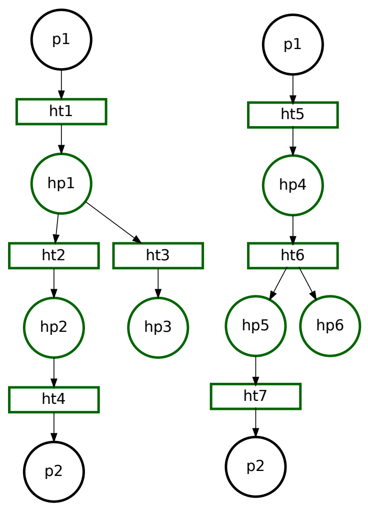
To determine the attachment points for the exchanged substructures, SC randomly selects two input places $p_b \in {}^\bullet t_\chi$ and $\bar{p}_b \in {}^\bullet \bar{t}_\chi$, and two output places $p_e \in t_\chi{}^\bullet$ and $\bar{p}_e \in \bar{t}_\chi^\bullet$ (Figure 7.6c); p_b, p_e, $\bar{p}_b$, and $\bar{p}_e$ will be used as "attachment" points for the incoming substructure, that is, the substructure identified by transition t_χ is attached to $\bar{\xi}$, in such a way that $\bar{p}_b$ is replaced by p_b, $\bar{p}_e$ is replaced by p_e, and vice versa (see Figure 7.6d).

Formally, the crossover mechanism acts on the sets of places, transitions and arcs of the RPN as follows:

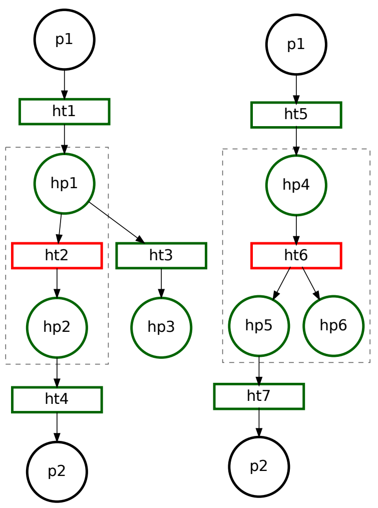
$$
\begin{aligned}
\bar{P}^h =& \bar{P}^h \cup P^+ \setminus \{\bar{p}_b, \bar{p}_e\};\\
P^h =& P^h \cup \bar{P}^+ \setminus \{p_b, p_e\};\\
\bar{T}^h =& \bar{T}^h \cup \{t_\chi\} \setminus \{\bar{t}_\chi\};\\
T^h =& T^h \cup \{\bar{t}_\chi\} \setminus \{t_\chi\};\\
\bar{F} =& \bar{F} \cup F^+ \setminus \bar{F}^+ \cup \{(\bar{t}, p_b) | \bar{t} \in {}^\bullet \bar{p}_b\} \cup \{(p_e, \bar{t}) | \bar{t} \in \bar{p}_e^\bullet\} \cup\\
& \cup \{(p_b, \bar{t}) | \bar{t} \in \bar{p}_b^\bullet\} \cup \{(\bar{t}, p_e) | \bar{t} \in {}^\bullet \bar{p}_e\} \setminus \{(\bar{t}, \bar{p}_b) | \bar{t} \in {}^\bullet \bar{p}_b\} \setminus\\
& \setminus \{(\bar{p}_e, \bar{t}) | \bar{t} \in \bar{p}_e^\bullet\} \setminus \{(\bar{p}_b, \bar{t}) | \bar{t} \in \bar{p}_b^\bullet\} \setminus \{(\bar{t}, \bar{p}_e) | \bar{t} \in {}^\bullet \bar{p}_e\};\\
F =& F \cup \bar{F}^+ \setminus F^+ \cup \{(t, \bar{p}_b) | t \in {}^\bullet p_b\} \cup \{(\bar{p}_e, t) | t \in p_e{}^\bullet\} \cup\\
& \cup \{(\bar{p}_b, t) | t \in p_b^\bullet\} \cup \{(t, \bar{p}_e) | t \in {}^\bullet p_e\} \setminus \{(t, p_b) | t \in {}^\bullet p_b\} \setminus\\
& \setminus \{(p_e, t) | t \in p_e{}^\bullet\} \setminus \{(p_b, t) | t \in p_b^\bullet\} \setminus \{(t, p_e) | t \in {}^\bullet p_e\}.
\end{aligned}
$$

The weights of the arcs exchanged during the crossover process are not modified. Only hidden places are moved between RPNs during the crossover process, because each RPN contains an identical set of fixed places, so that there is a biunivocal correspondence between the elements in P and P^h. For this reason, for each place $p \in ({}^\bullet t_\chi \cup t_\chi^\bullet) \cap P$, where t_χ is the transition selected for crossover, we let $\bar{F} = \bar{F} \cup \{(\bar{p}, t_\chi)\} \cup \{(t_\chi, \bar{p})\}$ and vice versa. It is important to clarify that the elements in P^h and T^h are "anonymous", that is, they do not share any semantics between different RPNs: if a hidden element is transferred from a RPN to another RPN during a crossover, it is considered a new unknown element with respect to the already existing elements.

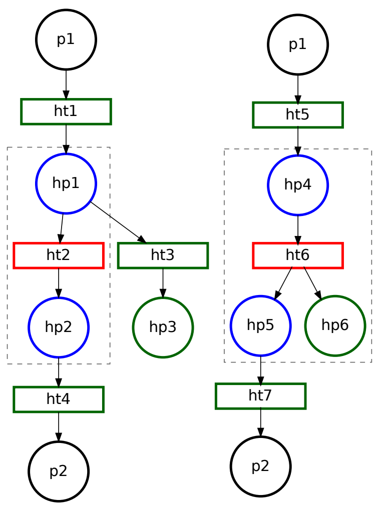
If the SC involves $p \in P^h$ and $\bar{p} \in \bar{P}$ as the selected "attachment" places of the crossover then, in the exchange of the substructure from ξ to $\bar{\xi}$, place $\bar{p} \in \bar{P}$ in $\bar{\xi}$



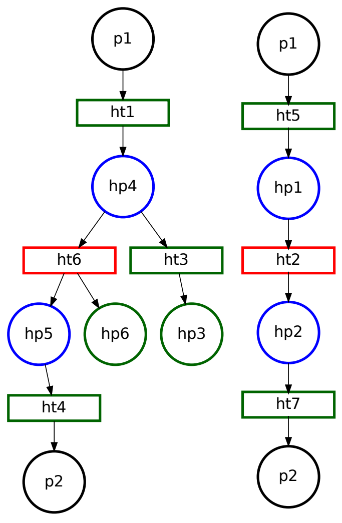
(a) Parents RPNs



(b) Step #1: selection of the transitions t_χ and $\bar{t}_\chi$

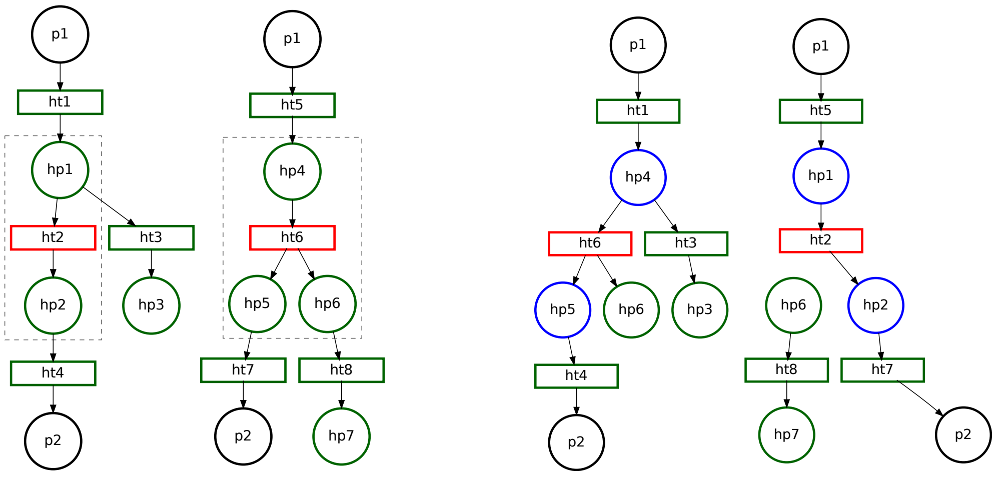


(c) Step #2: random selection of p_b, p_e, $\bar{p}_b$ and $\bar{p}_e$ nodes



(d) Step #3: exchange of substructures

Figure 7.6: Example of SC between two RPNs. (a) From each parent RPN (b) a hidden transition is randomly selected (red nodes ht_2 and ht_6), identifying the substructures that will be exchanged between the RPNs (dotted gray line). (c) One random place from the preset and one from the postset of ht_2 and ht_6 are selected (the blue nodes hp_1, hp_2, hp_4 and hp_5 that correspond to p_b, p_e, $\bar{p}_b$ and $\bar{p}_e$, respectively). (d) The substructures are exchanged between the RPNs and attached by replacing the respective blue nodes, thus yielding the two offspring. For the sake of compactness, the weights of arcs are not reported when they are equal to 1.



(a) Parents RPNs (b) RPNs after crossover

Figure 7.7: Example of SC breaking up one RPN, thus creating a separate component (the hidden transition ht_8 in Subfigure (b), and its pre- and postsets).

remains unchanged, while in the exchange from $\bar{\xi}$ to ξ, place $p \in P^h$ in ξ is replaced by $\bar{p}$. So doing, ξ will result in a consistent RPN, since $P = \bar{P}$.

SC is a convenient crossover operator for three reasons:

- it allows the crossover between two arbitrary RPNs, regardless the cardinality of the sets $P^h, \bar{P}^h$, which can vary during the genetic evolution;
- the pre- and post-order of the transitions of the offspring (as defined in Equations 7.3 and 7.4) are automatically conserved;
- the "directionality" of transitions is preserved, since SC swaps substructures whose presets are still presets, and elements of postsets are still postsets.

Nevertheless, the SC has two drawbacks:

- considering a substructure identified by a transition t_χ, if $\tilde{t} \neq t_\chi$ is connected to a place $p \in ({}^\bullet t_\chi \cup t_\chi^\bullet), p \notin \{p_e, p_b\}$, then the SC breaks the RPN leaving a separate component (see Figure 7.7);
- the SC allows the exchange of a single transition between two RPNs, which could have only marginal impact on large networks.

The second issue can be solved with two strategies: the first is to exploit some graph visiting algorithm (i.e., breadth- or depth-first) and extend the substructure accordingly; the second strategy is to determine a value $n_\chi \in \mathbb{N}$ such that $1 \leq n_\chi \leq \min\{|T^h|, |\bar{T}^h|\}$, and to repeat the crossover on n_χ different transitions, that is, to randomly create two vectors of indexes $i_1, \ldots, i_{n_\chi}$ and $j_1, \ldots, j_{n_\chi}$ (with $i_k, j_k \in \mathbb{N}$, $k = 1, \ldots, n_\chi$), and then apply the crossover operator on each couple $(t_{i_k}, \bar{t}_{j_k})$. The first strategy allows to exchange multiple transitions that were causally connected according to the chosen graph visiting algorithm, whilst the second one allows to exchange multiple independent transitions.

In the implementation of EPNs for real case applications, the computational complexity of applying the crossover operator is, in the worst case, $O(|F|^2 \cdot |P|)$. However, the use of a hash function might yield a reduction of the computational cost, in the average case, to $O(|P|)$.

The mutation operator

The mutation operator modifies the structure of a RPN $\xi(\mathfrak{g})$ in Ξ, or the properties of its places and arcs (i.e., capacity and weight), by acting on a single, randomly chosen hidden transition $t \in T^h$. In particular, the mutation operator associates $\xi(\mathfrak{g})$ to a new consistent RPN $\xi(\mathfrak{g}+1)$, according to a specified triple $\{p_{in}, t, p_{out}\}$. The rationale behind this triple is to provide new genetic material, that is, to modify the topology of the RPN; the functioning of the mutation operator for all the possible cases of $\{p_{in}, t, p_{out}\}$ is summarized in Table 7.2. After the application of any mutation case (except #8), $F = F \cup (p_{in}, t) \cup (t, p_{out})$ and $W(p_{in}, t) = W(t, p_{out}) = 1$. Cases #7 and #8 are particular and deserve a detailed explanation.

Case #7 introduces a brand new transition which is "disconnected" from the rest of the network, since it exploits new places that are not used by any other transition. So doing, the dynamics of these places, that is, the succession of their markings as a consequence of the firings, is independent from the rest of the RPN. In other words, Case #7 is a "silent" modification of the topology of the RPN, that will not have a direct impact on its behavior. Nevertheless, a further application of a genetic operator to the mutated RPN may connect this latent transition to the main net component, thus conditioning the behavior of the whole RPN.

Case #8 does not introduce any new genetic material. This operator is used to change the capacities $K(p)$ of randomly selected places $p \in P$, i.e. $K(p) = rnd$, where rnd is a random number sampled from the uniform distribution $(0, K_{\texttt{MAX}}]$ (where $K_{\texttt{MAX}}$ is the maximum capacity for the places in the RPN). Alternatively, the mutation can modify the weight of an arc; for instance, given the arc weight $W(p,t)$, its value can be updated as $W(p,t) = W(p,t) \pm 1$. It is worth noting that also Case #8 can lead to a modification in the structure of the RPN, whenever an arc weight is set to zero (i.e., the

Table 7.2: Effect of the mutation operator on a RPN $\xi(\mathfrak{g})$.

No.	*Condition*	$\xi(\mathfrak{g}+1)$	*Semantics of the mutation*
1.	$p_{in} \notin P \cup P^h$ $t \in T^h$ $p_{out} \in P \cup P^h$	$P^h = P^h \cup \{p_{in}\}$ $P^\infty = P^\infty \setminus \{p_{in}\}$	A new hidden place p_{in} is created and added to P^h; transition t is extended to have a new input place p_{in}
2.	$p_{in} \in P \cup P^h$ $t \in T^h$ $p_{out} \notin P \cup P^h$	$P^h = P^h \cup \{p_{out}\}$ $P^\infty = P^\infty \setminus \{p_{out}\}$	A new hidden place p_{out} is created and added to P^h; transition t is extended to have a new output place p_{out}
3.	$p_{in} \in P \cup P^h$ $t \notin T \cup T^h$ $p_{out} \in P \cup P^h$	$T^h = T^h \cup \{t\}$ $T^\infty = T^\infty \setminus \{t\}$	A new hidden transition t is created and added to T^h; transition t is connected to the existing input and output places p_{in} and p_{out}, respectively
4.	$p_{in} \notin P \cup P^h$ $t \notin T \cup T^h$ $p_{out} \in P \cup P^h$	$P^h = P^h \cup \{p_{in}\}$ $T^h = T^h \cup \{t\}$ $P^\infty = P^\infty \setminus \{p_{in}\}$ $T^\infty = T^\infty \setminus \{t\}$	A new hidden transition t is created and added to T^h; a new hidden place p_{in} is created and added to P^h; transition t is connected to new input place p_{in} and to an existing output place p_{out}
5.	$p_{in} \in P \cup P^h$ $t \notin T \cup T^h$ $p_{out} \notin P \cup P^h$	$T^h = T^h \cup \{t\}$ $P^h = P^h \cup \{p_{out}\}$ $P^\infty = P^\infty \setminus \{p_{out}\}$ $T^\infty = T^\infty \setminus \{t\}$	A new hidden transition t is created and added to T^h; a new hidden place p_{out} is created and added to P^h; transition t is connected to an existing input place p_{in} and to the new output place p_{out}
6.	$p_{in} \notin P \cup P^h$ $t \in T^h$ $p_{out} \notin P \cup P^h$	$P^h = P^h \cup \{p_{in}, p_{out}\}$ $P^\infty = P^\infty \setminus \{p_{in}, p_{out}\}$	Two new hidden places p_{in} and p_{out} are created and added to P^h; transition t is connected to p_{in} as input place and to p_{out} as output place
7.	$p_{in} \notin P \cup P^h$ $t \notin T \cup T^h$ $p_{out} \notin P \cup P^h$	$P^h = P^h \cup \{p_{in}, p_{out}\}$ $T^h = T^h \cup \{t\}$ $P^\infty = P^\infty \setminus \{p_{in}, p_{out}\}$ $T^\infty = T^\infty \setminus \{t\}$	A new hidden transition t is created and added to T^h; two new hidden places p_{in} and p_{out} are created and added to P^h; transition t is connected to the new input place p_{in} and to the new output place p_{out}
8.	$p_{in} \in P \cup P^h$ $t \in T^h$ $p_{out} \in P \cup P^h$	$P^h = P^h$ $T^h = T^h$ $P^\infty = P^\infty, T^\infty = T^\infty$	No new genetic material is introduced. Either p_{in} or p_{out} is randomly chosen; then, its capacity or the weight of the arc connecting it to t is modified

arc is removed). As a consequence, isolated hidden places or hidden transitions, which represent sources or sinks, can be introduced in the RPN after Case #8 is applied and must be removed. Therefore, after the application of this operator, the consistency of the RPN must be verified, regarding each place p and transition t involved in the mutation:

- if $p \in P^h$ and $\nexists \{(p,t),(t,p)\}\ \forall t \in T \cup T^h$, then $P^h = P^h \setminus \{p\}$;
- if $t \in T^h$ and $\nexists \{(t,p),(p,t)\}\ \forall p \in P \cup P^h$, then $T^h = T^h \setminus \{t\}$.

As a final step of the evolution process, pre- and post-order conditions of the offspring need to be verified, since the described mutations might produce a putative RPN whose topology does not satisfy Equations 7.3 and 7.4. In such a case, a further modification of the weights of the ingoing and/or outgoing arcs of a transition t is required. In particular, a randomly selected input (output, respectively) place of transition t is modified so that $W(\tilde{p},t) = W(\tilde{p},t) - 1$ (and/or $W(t,\tilde{p}) = W(t,\tilde{p}) - 1$), where $\tilde{p} \in {}^\bullet t$

($\tilde{p} \in t^\bullet$, respectively). It is clear that if $W(x, y) = 0$, then the corresponding arc (x, y) is removed from F. This operation is repeated until the RPN respects all the pre- and post-order conditions.

It is worth noting that these two mechanisms implicitly allow mutation to delete places, thus reducing the size of the RPN.

The computational complexity of the mutation operator is, for the worst case, $O(|P|)$. However, as in the case of crossover, the use of a hash function allows a reduction of the computational cost of this operator, in the average case, to $O(C)$ (where, in general, $C < |P|$).

7.2.2 Toward the application of EPNs for the RE of reaction-based models

In this section, the theoretical basis of a potential application of EPNs in the context of Systems Biology are sketched, in particular for the RE of biochemical interaction networks. As described at the beginning of this chapter, the RE problem consists in the identification of the network of reactions that describe the physical interactions among the chemical species occurring in the system. These networks can be defined thanks to human expertise, by relying on some pre-existing knowledge. However, most of the times the exact molecular mechanisms occurring in living cells are not known and cannot be fully understood by means of laboratory experiments only. This problem therefore demands the development of novel RE methods, so that a plausible network of biochemical reactions — able to reproduce some given experimental observations — can be determined in faster and inexpensive ways.

Here, it is briefly summarized the modeling of biochemical systems by means of PNs, and it is presented a possible strategy based on EPNs for solving the RE problem, which shows the feasibility of this methodology.

A biochemical network η can be modeled by means of the RBM formalism described in Section 2.2.1. Because of their bipartite graph structure, PNs offer an alternative and ideal conceptual framework for the modeling of such biochemical networks [53]. To this aim, the transformation of a network η into a corresponding PN (and vice versa) described in Section 2.2.5 can be exploited. According to those mappings, it is straightforward to show that the PN on the left in Figure 7.6a represents the following set of reactions:

- $R^h_1 : S_1 \rightarrow S^h_1$;
- $R^h_2 : S^h_1 \rightarrow S^h_2$;

- $R_3^h : S_1^h \rightarrow S_3^h$;
- $R_4^h : S_2^h \rightarrow S_2$.

The number of tokens (given as discrete or continuous value) in each place represents the molecular amount (given as number of molecules or concentration, respectively) of the corresponding chemical species, so that M_0 represents the initial state of the biochemical system.

The PN representation of a biochemical reaction network allows the investigation of its structural features by means of theoretical approaches [53, 345], like those described in Section 2.2.5. In addition, PNs can be easily extended with timed delays [181], or to incorporate quantitative information (e.g., reaction rates, concentration levels) to the aim of investigating the dynamic evolution of biochemical systems [110]. Kinetic parameters can also be associated to the reactions, in order to derive the probability of each reaction to occur [105], or to convert the system into a set of ODEs; in the latter case, places contain continuous values corresponding to the concentration of the chemical species associated to each place.

Once that a model of a biochemical network is defined in terms of a fully parameterized extended PN, according to the available domain knowledge, its dynamic behavior can be investigated. Anyway, when the domain knowledge is incomplete, uncertain or completely missing, a (fully parameterized) PN model of a biochemical system cannot be constructed. In such a case, it is necessary to perform the RE of the biochemical reaction network, investigating the unknown reactions and chemical species that are responsible for the observed phenomena.

As described in this chapter, many works perform the RE of biochemical systems by means of evolutionary techniques [44, 57, 143, 167, 178, 234, 241, 326, 336]. One limitation of many of these techniques is that the individuals are modeled by means of data structures that are not ideal to describe reaction networks; in addition, constraints and consistency controls (like in the case of cuRE) should be employed to control the quality and validity of the inferred network.

The strongest limitation of all these methods is that the cardinality of $\mathcal{S}$, that is, the number of chemical species that are present in the system, is assumed to be known and kept fixed during the optimization. This is generally a strong assumption, that may be justified only if the biochemical system is very well known (but, in such a case, the network should be known as well) or when laboratory experiments can yield this information with a certain precision. As a matter of fact, in most cases the exact number and nature of the chemical species involved in the system — including the intermediate complexes formed by the chemical bonds among various molecules — is unknown. Anyway, this is a fundamental information to properly carry out the RE of the system.

Thanks to the flexibility of hidden places and transitions, the use of RPNs to represent the candidate solutions gives to EPN the possibility to explore much more possibilities than the traditional RE approaches that exploit a fixed number of chemical species, thus leading to the formulation of new hypotheses for the structure of the biochemical network that should then be validated with *ad hoc* laboratory experiments. In this context, hidden places in the RPN can be exploited to represent some molecular species or complexes which are necessary to reproduce the expected behavior of the biochemical system, but that have not been yet identified with experimental techniques. Similar considerations hold for hidden transitions, which can represent molecular interactions that are not known from a biochemical point of view, but that might yield a better system functioning (in terms of the specified fitness function, and according to the available experimental data). A first attempt in this direction, which is based on CGP, is proposed in the next chapter for the automatic design of gene regulation networks.

The application of an EPN-based methodology for the RE of biochemical systems is similar to GP (Section 3.2.3). At first, a population $\mathcal{P}$ of RPNs is generated according to the available domain knowledge: when the information comes from established biological knowledge, the reactions are modeled using the sets P and T; on the contrary, when the information is uncertain, the network is modeled by means of hidden places and transitions. At generation $\mathfrak{g} = 0$, since the individuals are all identical, they undergo a preliminary mutation. During each generation, the best individuals are selected according to a specified fitness function, and the EPN genetic operators are applied to yield new offspring. The fitness function can be defined as in Equation 7.1, given that the hidden chemical species cannot represent a target.

To facilitate the evolutionary process and help the generation of biologically meaningful candidate solutions, in the use of EPNs for the RE problem it is possible to set $O_{pre} = 2$ to force the evolution of at most second-order reactions. This choice helps the EPN functioning, since it strongly reduces the search space Ξ, but at the same time it does not pose any limitation to the practical applicability of the RE methodology, as higher-order reactions can be mimicked through a cascade of consecutive reactions of lower order. During the evolutionary process, the EPN explores this search space, and eventually a best individual $\mathcal{I} \in \mathcal{P}$ emerges, representing a consistent RPN that fits with all the observed phenomena: $\mathcal{I}$ is finally returned as the result of the RE.

In what follows, the action of crossover operator for networks of biochemical reactions is sketched for the sake of clarity. Consider, for instance, the individuals ξ (left) and $\bar{\xi}$ (right) shown in Figure 7.6a: the crossover swaps the reactions R_2^h in ξ and R_6^h in $\bar{\xi}$, so that the chemical reaction R_2^h, renamed R_6^h after the crossover, has an additional product (i.e., S_6^h, left side of Figure 7.6d). On the contrary, the reaction R_6^h (renamed R_2^h after the crossover) in $\bar{\xi}$ loses a product and becomes a simple transformation from one species to another. The consequence of these modifications is that the new reaction network might have a completely different dynamic behavior, and hence a different (hopefully, better) fitness value.

The proposed fitness evaluation relies on the simulation of the candidate solutions, which is possible only if a proper parameterization of the candidate network in available. Hence, the RE problem is further complicated by the need of a PE methodology for the inference of the missing kinetic parameters [216, 232]: to this aim, an integrated EPN-based methodology embedding the PE process in each generation of the RE is under development.

A further difficulty of the RE process is due to the fact that different networks can lead to the same dynamic behavior; the problem of *indistinguishability* [327] — already mentioned in this chapter and discussed in Section 10.3 — cannot be solved without additional knowledge. EPNs do not directly mitigate this drawback, even though pre- and post-order conditions allow the reduction of the possible topologies, thus permitting the derivation of meaningful networks that can be then discriminated by domain expertise. Moreover, fixed places and transitions allow to easily introduce non-modifiable domain knowledge into the candidate solutions. Finally, the initial marking of the hidden places was set to zero: a further extension of EPN, in which the initial marking (i.e., state of the system) co-evolves with the topology, is under investigation.

Chapter 8

Evolutionary Design of synthetic networks

The goal of Synthetic Biology (SB) is the design and the construction of novel biological circuits — in particular, gene regulatory networks (GRNs) — able to reproduce a desired behavior. This task is similar to the RE problem [59] described in the previous chapter, whose purpose is to identify the network of interactions among the components of a real biological system, that fit an experimentally observed dynamics. In this context, mathematical models and computational analysis of GRNs can facilitate the experimental research and to provide useful insights for the control of gene interactions. The main difference between RE and the design of a synthetic circuit is that, in RE, the target dynamics is not a specifically chosen behavior, but it usually consists in laboratory measurements of some chemical species.

GRNs are traditionally modeled by means of high-level formalisms, in which the interaction of genes is expressed in terms of promotion and inhibition mechanisms. An example is represented by S-systems [297], namely, systems of non-linear Ordinary Differential Equations (ODEs) in which gene expression is modeled by power-law functions (see Section 2.2.4). S-systems models of GRNs are able to capture the intrinsic non-linearity of gene expression, providing a good description of the behavior of the corresponding artificial gene regulation system [239]. In the context of SB, the inference of S-systems was tackled by means of Evolutionary Computation (EC) [13, 57, 143, 239], where a population of candidate solutions iteratively evolves under the pressure of a fitness function. The EC technique traditionally applied to this problem is Genetic Programming (GP) [168], in which candidate GRNs are encoded by complex recursive data structures like derivation trees or LISP s-expressions. Another EC methodology that was exploited for the inference of S-systems is Differential

Evolution [324]; however, this method suffers from the additional problem related to overfitting solutions, that must be tackled during the evolution of the typical sparse structure of GRNs.

S-systems provide a valuable formalism for the modeling of GRNs, but they are not powerful enough to describe the actual mechanisms allowing a GRN to express a certain dynamics, so that these models have a low predictive capability. In order to provide biologists with additional information regarding gene regulation mechanisms, GRNs can be modeled as mechanistic reaction-based models (RBM), which describe in detail the molecular interactions among the chemical species. An advantage of using RBMs is that they can be easily exploited to analyze and predict emerging dynamics of GRNs in different conditions, thanks to several existing simulation tools [140, 233, 235].

In this chapter, it is called *Evolutionary Design* (ED) the problem of automatically deriving a RBM able to reproduce a desired behavior, by exploiting EC methods only. The ED of RBMs can be more complicated than the ED of S-systems, because RBMs also require a proper description of the stoichiometry of reagents and products in each biochemical reaction. Furthermore, as in the case of S-systems, a correct kinetic parameterization of all reactions is needed, in order to produce a reliable simulation of the system dynamics. As described in Section 6.1, the PE problem can be tackled by means of EC as well and, in particular, using PSO (introduced in Section 3.3.2).

In this chapter, it is presented a computational strategy to infer a RBM that specifically represents a Gene Regulation Model (GRM) characterized by a predefined behavior. In particular, the ED methodology, named cuGENED, integrates two EC algorithms: CGP [214] and PSO [157], described in detail in Section 3.2. CGP exploits individuals encoded as fixed-length vectors of integer numbers which, in contrast to standard GP, are mapped onto directed graphs rather than derivation trees [214].

As in the case of the cuRE algorithm (Chapter 7), the choice of the CGP for the ED process is motivated by the fact that graphs are suitable for the representation of a network, and the mapping between a CGP individual and the corresponding GRM allows a direct translation of the individual into a human-comprehensible set of chemical reactions. Moreover, since CGP exploits fixed-length individuals, it does not need any explicit strategy to avoid bloating, that is, the uncontrolled growth of the size of solutions that usually occurs in GP. Moreover, cuGENED inherits all the closure and sufficiency properties characterizing cuRE.

cuGENED exploits CGP to derive the network of biochemical reactions which describe the genetic regulation mechanisms in a GRM. Then, PSO is exploited to estimate the kinetic parameters of the GRMs. The target of the overall evolutionary process is represented by some representative dynamics of a predefined number of genes that participate in the GRN. Eventually, the effective behavior of the inferred model has to be validated after the synthetic engineering of the genetic circuit. Such validation might be done by measuring, e.g., the transcription levels of the genetic components of the circuit.

It is worth noting that, for the ED of synthetic GRNs, it is not possible to consider any network structure as target of the optimization process, so that the quality of a candidate GRM can be only evaluated by comparing its simulated dynamics against the desired target behavior. Indeed, an absolute novelty of cuGENED with respect to the state-of-the-art, is that it allows **0-knowledge inference**, since the number of intermediate chemical species occurring in the GRM is usually unknown *a priori*.

In cuGENED, every generation of CGP requires the execution of PSO, that realizes the PE for each individual. Since PSO is a population-based algorithm as well, the whole methodology is computationally expensive. Nevertheless, all fitness evaluations in each iteration of the PSO are independent and can be accelerated by means of a parallel architecture. To this purpose, cuGENED exploits cupSODA [233], the GPU-powered deterministic simulator of biochemical systems modeled by means of RBMs introduced in Section 5.1. cupSODA, exploited during the PE phase, is fully integrated in cuGENED and allows a strong reduction of the overall running time.

The main contents of this chapter will appear in the forthcoming book *Evolutionary Algorithms in Gene Regulatory Network Research*, edited by H. Iba and N. Noman and published by John Wiley & Sons [236].

8.1 RBMs of gene regulation

Given a biochemical system η consisting of some molecular species (e.g., genes, proteins, metabolites) and their mutual interactions, a mechanistic reaction-based model (RBM) of η can be formally defined as described in Section 2.2. In this chapter, at most second-order reactions — that is, reactions that have no more than two reactant molecules — are considered in RBMs, since third-order (or any higher order) reactions have a probability to occur almost equal to zero, as they would require the simultaneous collision of three (or more) reactant molecules.

This chapter focuses on reaction-based gene regulation models (GRMs), which formally describe the biochemical reactions involved in gene transcription and translation. To properly define these reactions, it is assumed that the set $\mathcal{S}$ of molecular species is given by the union of two disjoint sets of species, which are denoted by

$\Gamma = \{\gamma_0, \ldots, \gamma_{N_\Gamma}\}$ and $\Sigma = \{\sigma_0, \ldots, \sigma_{N_\Sigma}\}$, for some $N_\Gamma, N_\Sigma \in \mathbb{N}$. It is assumed that $|\Gamma| \leq |\Sigma|$, where $|\cdot|$ represents the cardinality of the set.

The set Γ represents the genes that are available for the synthetic engineering of gene regulation circuits. Each element γ_i in Γ is strictly associated to a messenger RNA (mRNA), the product of gene transcription, whose expression can be experimentally evaluated through cutting edge technologies (e.g., microarray [98], qRT-PCR [170]). Thus, the cardinality of Γ is equivalent to the number of target time-series used in cuGENED. By abuse of notation, γ_i will identify either the "gene" or its "transcription product", according to the context.

The set Σ represents a set of generic species, as well as their mutual chemical complexes, that are related to the processes of gene expression. These species are the actual effectors of gene regulation, and need to be included in any GRM to evaluate their influence on the emerging behavior of the circuit. An element $\sigma_i \in \Sigma$ might represent any kind of gene product (e.g., protein) related to γ_i, or the molecular complex formed by the interaction of the gene product (acting as promoting/inhibiting transcription factor) with another gene $\gamma_j \in \Gamma$.

To better clarify the meaning of the sets Γ and Σ, some types of reactions that formally describe the processes of gene regulation, and that might be present in a GRM, are illustrated along with other generic reactions mentioned above.

Reactions that represent gene expression (transcription/translation) can be written in the form $\gamma_i \rightarrow \gamma_i + \sigma_i$. Given a gene $\gamma_j \in \Gamma$, its regulation by means of gene γ_i can be described either as $\gamma_i + \gamma_j \rightarrow \sigma_k$ or as $\sigma_i + \gamma_j \rightarrow \sigma_k$, with i not necessarily distinct from j. Here, the species σ_k might represent two different elements: 1) the product of the regulation of γ_j (e.g., the protein encoded by γ_j), in this case we can simply say that σ_k is equal to σ_j; 2) the chemical complex between gene γ_j and its own regulator, in this case σ_k is a compact formalization for the species $\gamma_i\gamma_j$ or $\sigma_i\gamma_j$. The inverse reactions of the type $\sigma_k \rightarrow \sigma_i + \gamma_j$ can then be used to describe the dissociation of the regulator σ_i from gene γ_j. The effective expression of a regulated gene γ_j — that takes place after the occurrence of some reactions of type $\sigma_i + \gamma_j \rightarrow \sigma_k$ — can be synthetically written in the form $\sigma_k \rightarrow \sigma_j$. This reaction states that species σ_j, related to gene γ_j, is derived from species σ_k, which represents an intermediate molecular complex having γ_j as element (e.g., $\sigma_k = \gamma_i\gamma_j$ or $\sigma_k = \sigma_i\gamma_j$, as mentioned above). Finally, reactions of the form $\sigma_h \rightarrow \sigma_g$ can also be used to represent the degradation, or any other generic transformation reaction (e.g., post-translational modification of a protein) of the species occurring in Σ.

As it was shown in the previous chapter, the RE process of a given biological system consists in the automatic identification of the network of interactions among the molecular components of that system. In this task, a set of experimental time-series measurements of some species occurring in the system can be exploited as target to drive the evolutionary inference of the network [234]. On the contrary, in this chapter it is assumed that only an expected dynamical behavior of the network is given as target. Quite obviously, it is also assumed that no specific laboratory measurements of the temporal evolution of chemicals that will constitute the circuit can be given *before* the circuit itself has been engineered. For this reason, in what follows it is not formally specified the type and the unit measurements of the amount of chemicals, but the general term of "level" is used. The level can indicate either the number of molecules or the concentration values that might then be actually measured by *ad hoc* laboratory methodologies *after* the construction of the circuit. For instance, if qRT-PCR is exploited to validate the functioning of the synthetically engineered circuit, then the level of the species in Γ represents the concentrations of the corresponding mRNA, measured by relative quantification with respect to the amount of some control housekeeping gene [192].

According to the law of mass-action, given a set of biochemical reactions $\mathcal{R}$ — each one characterized by its own kinetic constant k_j[1] — for each molecular species appearing either as reagent or product in some reaction in $\mathcal{R}$, it is possible to derive a rate equation that describes the variation in time of its concentration. In other words, any given RBM or GRM can be formalized as an equivalent system of coupled (non-linear) first-order ODEs [358]. This formalization can be exploited to determine the temporal evolution of the network (that is, its dynamics) by means of different numerical integration methods [41]. These algorithms require as input the set of ODEs, along with the set of kinetic constants and the initial concentrations of the chemical species. To this aim, in this chapter it is exploited cupSODA [230, 233], which automatically derives a set of ODEs from RBMs defined according to MAK, and then exploits the numerical integration algorithm LSODA [257] to perform the deterministic simulation of the system dynamics. In Section 8.2 it is shown how to exploit CGP to carry out the ED of a GRM, which is then automatically converted to an ODEs system and simulated by means of cupSODA, considering the kinetic constants that are inferred by PSO.

[1]The unit of measurement of k_j, that I omit for simplicity, depends on the order of each reaction: it is equal to $(\text{mol/L})^{1-n}(\text{time})^{-1}$ for reactions of order n, $n \geq 1$, and equal to $(\text{time})^{-1}$ if $n = 0$.

In this chapter, candidate solutions of CGP evolve by using a (1+λ) ES (see Section 3.2.2), as described in [214]: all individuals are evaluated and the best one is selected as a parent for the next generation. Then, λ offspring are produced by means of random mutations, that is, random modifications of the integers which constitute the genotype of the parent individual. The proportion of genes that are mutated is determined by the mutation rate parameter $\rho \in (0, 1)$. The ES methodology does not exploit any crossover mechanism.

In Section 8.2 it is shown how to exploit CGP to perform the ED of GRMs. In order to evaluate the GRM that each CP represents, a proper fitness function must be defined. As target of the optimization process, it is exploited a set of artificial temporal data that represent the desired behavior of the synthetic gene regulation circuit. These artificial data informally correspond to the concentrations of the molecular species that might be experimentally measured after the construction of the circuit itself. Therefore, the fitness function is based on the comparison of the target series against a simulated dynamics of the model, which is generated by exploiting the cupSODA tool [230, 233]. Since the GRM derived with CGP is not complete until a proper kinetic parameterization is given, the fitness evaluation of each candidate solution embeds a PE phase, which is performed by means of PSO.

In this chapter, a particle corresponds to a candidate kinetic parameterization of each GRM determined by CGP. Therefore, M is equal to the number of connected outputs in the CP, i.e., the total number of reactions in the GRM.

8.2 ED of GRMs by means of CGP and PSO

In this section, the ED methodology is presented. cuGENED is based on the integration of 1) CGP for the design of the synthetic circuit, 2) PSO for the estimation of the kinetic parameters of the corresponding GRM, and 3) cupSODA for the GPU-accelerated fitness evaluation.

To the best of my knowledge, this is the first time that the ED is performed by means of a hybrid approach based on CGP and PSO. The choice of PSO to perform the PE task is motivated by empirical studies in the field, highlighting its better performances with respect to other popular global optimization techniques like GAs, Differential Evolution and Evolution Strategy [26, 78].

CGP phase. In cuGENED, CGP is implemented using $\mathfrak{F} = \{+, -\}$ as the set of functions. The composition of input nodes and functional nodes allows CGP to encode complex expressions for each genotype G. The expressions can be then converted, by means of symbolic manipulation, into arithmetical equations as described hereafter.

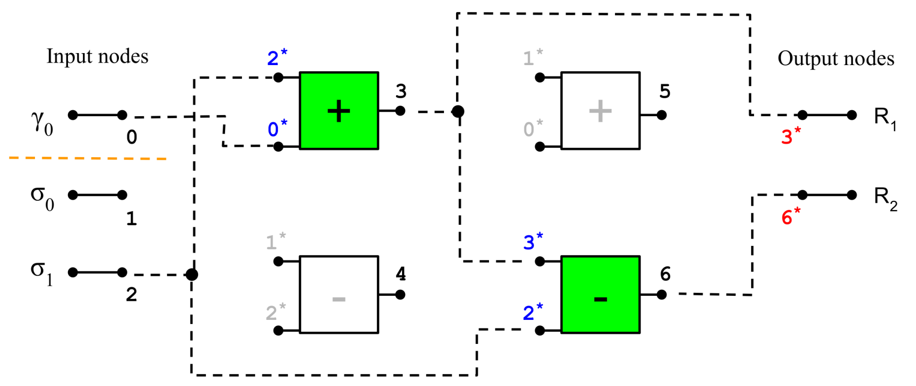
In the context of ED of GRMs, the set of input nodes is determined by the species occurring in Γ and Σ, so that $n_i = |\Gamma| + |\Sigma|$. Since the exact regulatory interactions necessary to reproduce the desired behavior of the synthetic circuit are to be determined, it follows that the number of chemical species belonging to the set Σ is unknown as well. Therefore, also the cardinality of Σ must be inferred by means of some heuristics. As a matter of fact, the value $|\Sigma|$ has a relevant impact on the ED process, since it determines the space of the possible GRMs that CGP can explore. If $|\Sigma|$ is too small, then the circuit able to achieve a perfect fit of the desired dynamics might be impossible to be designed. On the contrary, if $|\Sigma|$ is too large, then the evolutionary algorithm can take a longer time to converge to an optimal solution, or might suffer from over-fitting. Nevertheless, in the second case, CGP is able to automatically exclude the unnecessary chemical species from the set of candidate solutions if they have no impact on the fitness value. In this work, it is always used the value $|\Sigma| = 4 \cdot |\Gamma|$ as heuristic, because it is assumed that, for each target species in Γ, four intermediates are enough to model gene regulation mechanisms.

The number of input connections for each functional node is $n_n = 2$, because only addition and subtraction are exploited as functions for the construction of the expressions. The set of output nodes corresponds to the biochemical reactions that will be part of the inferred GRM. In these reactions, the species corresponding to the input nodes appear either as reactants, if a functional node containing function $+$ is crossed, or as products, if a functional node containing function $-$ is crossed. So doing, each expression encodes a single reaction of the GRM.

Generally, in CGP the number of output nodes n_o is fixed. In the context of GRMs, this implicitly means that the actual number of reactions should be known before the optimization takes place, which is clearly unreasonable. Since the GRM itself is the goal of the optimization, in this chapter the value n_o represents an upper bound to the number of reactions that will appear in the model. Hence, a proper choice for n_o is fundamental: it should be large enough to include all the necessary regulatory mechanisms, and small enough to avoid bloating and over-fitting. As heuristic for the selection of n_o, its value could be fixed equal to the number of all possible reactions that can involve the set of genes and the set of generic chemical species, i.e., $n_o = |\Gamma| \cdot |\Sigma| + |\Gamma| + |\Sigma|$. Furthermore, in this implementation the value of l is set to n_c, meaning that output nodes can be connected to any functional or input node, and each functional node belonging to column j can be connected only to nodes between column $j-1$ and the input nodes. An example of CP related to the ED problem of GRMs is described in Example 8.2.1.

Example 8.2.1 *There exists a strict correspondence between the genotype of the CP and the connectivity of each node in the Cartesian coordinate grid. Functions + and − in the grid are represented by 0 and 1 in the genotype, respectively (the first number in each of the four triplets). Since + and − are binary operators, $n_n = 2$ input connections are used to each functional node. The grid of functional nodes consists in $|FN| = n_r n_c = 4$ nodes, since the grid is composed of two rows and two columns. The genotype is generated by randomly drawing values related to the chemical species (0 for γ_0, 1 for σ_0, 2 for σ_1) and to the output of each functional node (3, …, 6). Then, these values are assigned to the input of the functional nodes and to the output nodes (numerical values with asterisk). Note that, in the construction of the genotype, given a functional node only the values of previous input and functional nodes can be randomly selected and assigned to its input connections, since $l = n_c$. The grid is parsed backwards, from the $n_o = 2$ output nodes R_1, R_2 to the $n_i = 3$ input nodes $\gamma_0, \sigma_0, \sigma_1$. The resulting equations are then automatically converted into an equivalent set of biochemical reactions. The green functional nodes are involved in the equations, since a path from the input nodes to the output nodes exists here. The white functional nodes, instead, are not connected in the grid and do not participate in any equation in this CP. Thus, their corresponding genes (gray numbers) are non-coding sequences.*

CP Genotype 0 2 0 1 1 2 0 1 0 1 3 2 3 6



Derived equations $R_1: \gamma_0 + \sigma_1$
$R_2: (\gamma_0 + \sigma_1) - \sigma_1$

Corresponding reactions
$$\left.\begin{array}{l} R_1 : \gamma_0 + \sigma_1 \rightarrow \emptyset \\ R_2 : \gamma_0 + \sigma_1 \rightarrow \sigma_1 \end{array}\right\} \eta$$

The pseudocode of the ED methodology based on CGP and PSO is reported in Algorithm 7.1. The evolutionary process begins with the creation of a population of $I = 1 + \lambda$ random CPs (lines 2–7). The population evolves by means of an ES process which can be decomposed in three main steps:

1. in **Step 1**, the representation of each CP is converted into a GRM. Then, PSO is used to estimate the values of the kinetic constants and to assess the fitness value of the parameterized candidate network (lines 10–14);

2. in **Step 2**, the CPs are ranked according to the fitness values, to identify the best CP in the population (line 15);

3. in **Step 3**, a brand new population is formed by considering the best CP, together with the $I - 1$ offspring created by applying the mutation operator on the best CP (lines 16–30).

The figure in Example 8.2.1 schematizes the conversion of a CP into the corresponding GRM (**Step 1**). The connections of each output node are followed backwards by recursively passing through the functional nodes. This process yields a derivation tree that is translated into an arithmetical equation, where positive terms (respectively, negative) are considered as reactants (products). Each equation produces a single candidate chemical reaction for the network. A candidate GRM η for the synthetic circuit is then obtained by repeating the algorithm for all output nodes.

The fitness values of all candidate solutions are calculated at the end of the PE process by means of PSO, so that a ranking of the CPs can be assessed and the best solution ($best_{CP}$) in the CGP population is identified (**Step 2**).

During each generation of the CGP, the new offspring solutions are obtained by applying a mutation operator to $best_{CP}$. The GRM η is compared to the rest of the population: if it is identical to some other CP, then it is mutated again in order to achieve a heterogeneous population, and the process is repeated. In this methodology it is not verified that η consists of a single connected component, since not all chemical species in Σ necessarily need to be present in the GRM. The rationale behind this choice is that, in the ED methodology presented here, it is not specified *a priori* the exact number of chemical species that should occur in the system (it is only provided an upper bound). Therefore, in this methodology also the number of chemical species in Σ undergoes the optimization process. This approach is different from typical methods used for the RE problem. In particular, it differs from the RE method based on CGP and PSO previously proposed in [234]: in that case, the number of chemical species

```
1: $CGP\_population \leftarrow$ create_empty_population()
2: for 1 to I do
3:     $CP =$ create_CP($\mathcal{S}, n_0, n_r, n_c$)
4:     $CGP\_population$.add($CP$)
5: end for
6: for 1 to $CGP\_generations$ do
7:     for all $CP$ in $CGP\_population$ do
8:         $network \leftarrow CP$.convert_to_network()
9:         $CP.fitness, CP.parameters \leftarrow$ PSO($network$)
10:    end for
11:    $best_{CP} \leftarrow$ find_best_individual($CGP\_population$)
12:    $CGP\_population \leftarrow$ create_empty_population()
13:    $CGP\_population$.add($best_{CP}$)
14:    for 1 to $I - 1$ do
15:        repeat
16:            $candidate_{CP} \leftarrow best_{CP}$.mutation($\rho$)
17:        until $candidate_{CP}$ not in $CGP\_population$
18:        for all $reaction$ in $candidate_{CP}$ do
19:            if not consistent($reaction$) then
20:                $candidate_{CP}$.remove($reaction$)
21:            end if
22:        end for
23:        $CGP\_population$.add($candidate_{CP}$)
24:    end for
25:    if fitness($elite_{CP}$) $<$ fitness($best_{CP}$) then
26:        $elite_{CP} \leftarrow best_{CP}$
27:    end if
28:    $CGP\_population$.add($elite_{CP}$)
29: end for
30: return $bestCP$
```

Figure 8.1: Pseudocode of the GRM design algorithm

participating in the system was known *a priori*, therefore it was mandatory for the candidate solutions to consist in a single connected component.

When a new population of CGP is generated, for each candidate solution a consistency check is performed (lines 24–28 in Algorithm 7.1). Specifically, it is verified that each reaction in the GRM η obeys the conditions described in Section 2.2.1, together with the additional condition that no identical reactions should appear in η. In particular, since at most second-order reactions are considered, the length of CP expressions can be limited to reduce the bloating by setting $n_c \leq 3$. In general, if a reaction R is not consistent, it is removed from the network, so that $\eta = \eta \setminus \{R\}$. Eventually, when a new consistent CP is produced, the individual is inserted into the new population.

To improve the convergence speed, at the end of **Step 3** the elitist selection is applied by adding a non-mutated copy of the best candidate network found so far. This solution becomes the $(I+1)$-th individual of the population (lines 31–33 in Algorithm 7.1). When the new population is formed, the fitness value of each individual is calculated by executing a PE by means of PSO, as described hereby. This iterative process is repeated for $GEN_{\texttt{MAX}}$ generations. For the tests presented in this chapter, $GEN_{\texttt{MAX}} = 100$. Finally, the GRM with the best fitness, along with its parameterization, is chosen as the result of the ED problem.

PSO phase. For the execution of the PE task, each particle in the PSO corresponds to a candidate kinetic parameterization of a GRM inferred by CGP. Namely, for each candidate GRM, a whole swarm of particles is exploited to determine the reaction constants that best fit the desired target behavior for that GRM. To this aim, the PSO adopts the following settings: $n = 64$ particles, randomly generated using a logarithmic distribution to better distribute the values of kinetic constants over different orders of magnitude; $C_{soc} = C_{cog} = 2.05$, as suggested in [78]; inertia linearly decremented from $w = 0.9$ to $w = 0.4$; velocity vector automatically clamped to a maximum intensity, equal to 10% of the maximum point-to-point distance in the search space; search space bounded between 10^{-5} and 10^{2} (for each kinetic parameter) with damping boundary conditions [361].

The choice of the upper bound of the search space for the kinetic constants is fundamental for the entire optimization process, since a limited range of variation might exclude the global optimum. However, to perform PE of biochemical systems consisting in a small set of reactions, and characterized by molecular species with low concentrations, limiting the search space of parameters is a good practice for the following reason. Very high values of kinetic constants could lead to undesired dynamics in which chemical species are entirely consumed in the first time instants of the simulation.

The movement of the swarm during the PE phase is determined according to the fitness function. In this chapter, for each particle ϕ_i, $i = 1, \ldots, n$, which encodes the kinetic parameterization of a GRM η, the fitness is defined as the normalized distance between the values of the simulated dynamics of gene expression levels in η and the desired target dynamics. Formally, given the set Γ of genes that are available for the synthetic engineering of the gene regulation circuit and the set $\mathcal{R}$ of reactions of the GRM inferred by CGP, $X_h(t_c)$ denotes the expected measurement at time t_c of the h-th chemical species in Γ, where $Y_h^{\phi_i}(t_c)$ denotes its simulated value sampled at time t_c. The value $Y_h^{\phi_i}(t_c)$ is obtained by means of cupSODA, using the kinetic constants contained in particle ϕ_i. The fitness function $\mathcal{F}$ of particle ϕ_i is therefore given by:

$$\mathcal{F}(\phi_i) = \sum_{c=1}^{C} \sum_{h=0}^{N_\Gamma} \frac{|Y_h^{\phi_i}(t_c) - X_h(t_c)|}{X_h(t_c)}, \tag{8.1}$$

where $C \in \mathbb{N}$ corresponds to the number of time instants that are arbitrarily sampled in the target dynamics of the GRM.

In the tests presented in this chapter, the PSO algorithm is halted after $IT_{\texttt{MAX}} = 300$ iterations. Then, for each GRM η, the value $\mathcal{F}(\mathbf{g}_\eta)$ — that is, the fitness of the best solution $\mathbf{g} = (k_1, \ldots, k_M)$ determined within the swarm of all candidate parameterizations of η — is taken as the fitness of the inferred GRM η. The GRM characterized by the minimum fitness value among all GRMs inferred by CGP, is eventually chosen as the best solution of the whole ED problem.

GPU implementation. Fitness evaluations are computationally expensive, since multiple simulations (one for each particle) must be performed during all iterations of the PE on each candidate network inferred by CGP. More precisely, it takes $O(GEN_{\texttt{MAX}} \cdot IT_{\texttt{MAX}} \cdot I \cdot n)$ fitness evaluations to perform a whole ED process. Anyway, all simulations during each PE phase are mutually independent and can be straightforwardly accelerated by means of a parallel architecture. In order to take advantage of the parallelism of modern GPUs, the GPU-accelerated simulator cupSODA [230, 233] is exploited to launch n threads, which perform the simulations and calculate the fitness functions defined in Equation 8.1. This way, the impact of fitness evaluations on the overall running time decreases to $O(GEN_{\texttt{MAX}} \cdot IT_{\texttt{MAX}} \cdot I)$.

The rest of the methodology (CGP and PSO) is implemented using the Python language (version 2.7) and is executed in a strictly sequential fashion. PSO invokes cupSODA to assess the fitness values by means of synchronous subprocess calls.

8.3 Results

To test the feasibility and the effectiveness of cuGENED, the ED of synthetic circuits composed by two and three genes is performed, using as target of the optimization process a desired temporal dynamics of a small set of (*in silico* generated) mRNA levels. The target time-series were created from scratch to be complex enough to require the ED of GRMs possibly characterized by multiple regulation mechanisms and intermediate species.

As previously mentioned, the results of the ED process can be only evaluated by considering the desired dynamics. No target GRM can be exploited to assess the quality of the inferred network, since the goal of the ED process is to determine a synthetic circuit that still has to be constructed in laboratory. Therefore, to prove the effectiveness of cuGENED, only comparisons between the expected and the simulated dynamics of the inferred GRM are presented hereby.

The following tests were performed on a workstation with a CPU Intel Core i7-2600, clock frequency of 3.4 GHz, and with a GPU Nvidia GeForce GTX 590, running OS Windows 7 64 bit. The settings of the LSODA integrator used for the simulation of the dynamics of the candidate solutions are: relative error equal to $1 \cdot 10^{-10}$, absolute error equal to $1 \cdot 10^{-10}$, maximum number of integration steps equal to 10000.

8.3.1 ED of synthetic circuits with two genes

For the ED of synthetic circuits consisting of two genes, $|\Gamma|$ and $|\Sigma|$ were set to 2 and 8, respectively, so that the inferred GRM can contain at most $n_i = 10$ chemical species and $n_o = 26$ reactions (according to the heuristics described in Section 8.2). The desired target dynamics of the species in Γ are described by two sigmoidal curves, characterized by different slopes. This behavior corresponds to a down-regulation of both genes, whose constitutive expression decreases in time by the action of some mutual regulatory mechanism, that has to be inferred by cuGENED.

Some preliminary tests were executed to analyze the influence of cuGENED settings on the ED process. In the first test, the impact of the parameter n_r — that is, the number of rows in the grid of functional nodes — on the convergence speed was investigated (Figure 8.2). According to results for the two-genes system under investigation, too small ($n_r = 8$) or too large ($n_r = 14$) values of n_r yield a worse convergence of cuGENED, while the best results are achieved with $n_r = 10$. Moreover, by analyzing the early phases of the CGP process, one can observe that the higher the n_r value the faster the convergence. Nevertheless, it is the intermediate value $n_r = 10$ that achieves the best convergence speed, confirming a similar result presented in [234]. In addition, Figure 8.2 highlights a feature that is typical of methodologies embedding PSO into CGP, as previously discussed in [234]: due to the stochasticity of PSO, two different PE executions on the same network usually lead to two different kinetic parameterizations which, in turn, are likely to have two different fitness values, explaining the large fluctuations in the convergence speed plots.

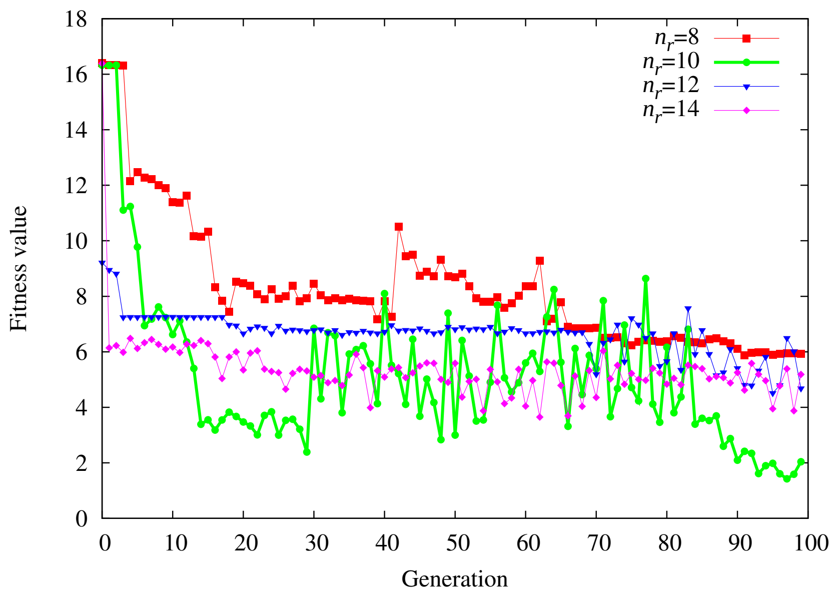


Figure 8.2: Fitness value of the best solution of the ED process obtained by varying the settings for n_r in CGP. The best setting for the two-genes system under investigation is $n_r = 10$, while smaller and larger values yield worse results. The plot highlights the presence of fluctuations due to the stochasticity of PSO, emphasizing how a proper choice for the grid size is fundamental for the convergence of cuGENED.

As a second test, it was analyzed the impact of the parameter ρ, which determines how many elements of a CP genotype are mutated during the offspring generation. Figure 8.3 shows the convergence of cuGENED with different values of ρ, highlighting that the best choice is $\rho = 0.2$. This setting represents the best trade-off between exploration and exploitation of the search space. Indeed, if the value of ρ is too small, CGP is not able to properly explore the search space and cannot converge to individuals characterized by a good fitness value. On the contrary, if the value of ρ is too large,

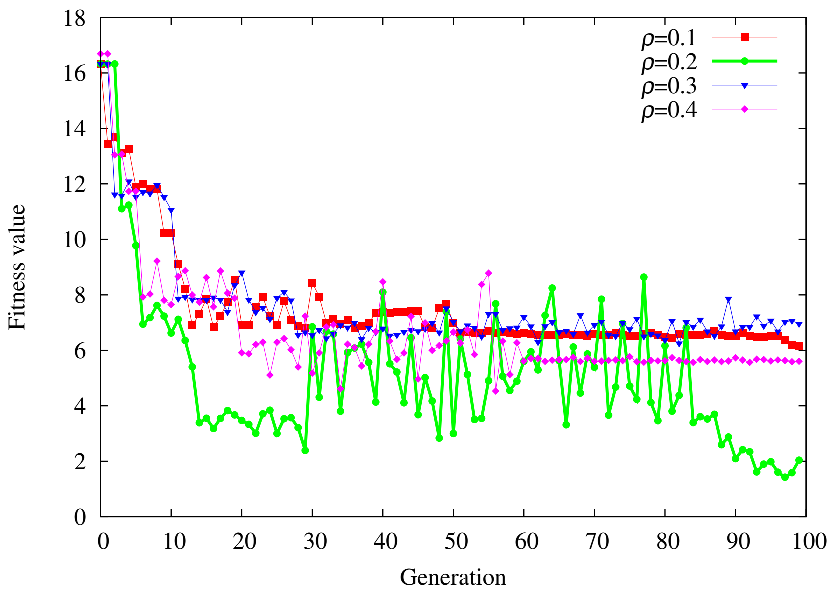


Figure 8.3: Fitness value of the best solution of the ED process obtained by varying the settings for ρ in CGP. The best setting for the two-genes system under investigation is $\rho = 0.2$, that is, about 20% of the genome must be modified when producing new offspring during each generation of the CGP.

the mutation easily disrupts the structure of the best individual, yielding random individuals with a large fitness value and reducing the advantage of an evolution-guided exploration of the search space.

The best setting identified for cuGENED (i.e., $n_r = 10$ and $\rho = 0.2$) was then exploited to derive the GRM $\eta_{2\gamma}$ that achieves the best fit with the expected behavior of the two-genes system. Figure 8.4 shows the comparison between the desired target dynamics (dots) and the simulated dynamics (lines) of $\eta_{2\gamma}$. This network is represented in Figure 8.5 and consists of the following reactions:

$$\begin{aligned}
R_1 &: \gamma_1 + \sigma_3 \xrightarrow{27.535} \sigma_2,\\
R_2 &: \gamma_1 + \gamma_0 \xrightarrow{2.390} \sigma_1,\\
R_3 &: \sigma_2 + \sigma_4 \xrightarrow{13.753} \sigma_3,\\
R_4 &: \sigma_2 \xrightarrow{19.131} \sigma_3,\\
R_5 &: \sigma_2 \xrightarrow{7.697} \sigma_4,\\
R_6 &: \gamma_1 \xrightarrow{10.014} \sigma_2 + \sigma_1 + \gamma_1,\\
R_7 &: \sigma_2 + \gamma_0 \xrightarrow{2.303} \sigma_4,\\
R_8 &: \sigma_3 + \gamma_0 \xrightarrow{15.420} \sigma_4,
\end{aligned}$$

where the numerical values above the arrows correspond to the best kinetic parameterization determined by PSO.

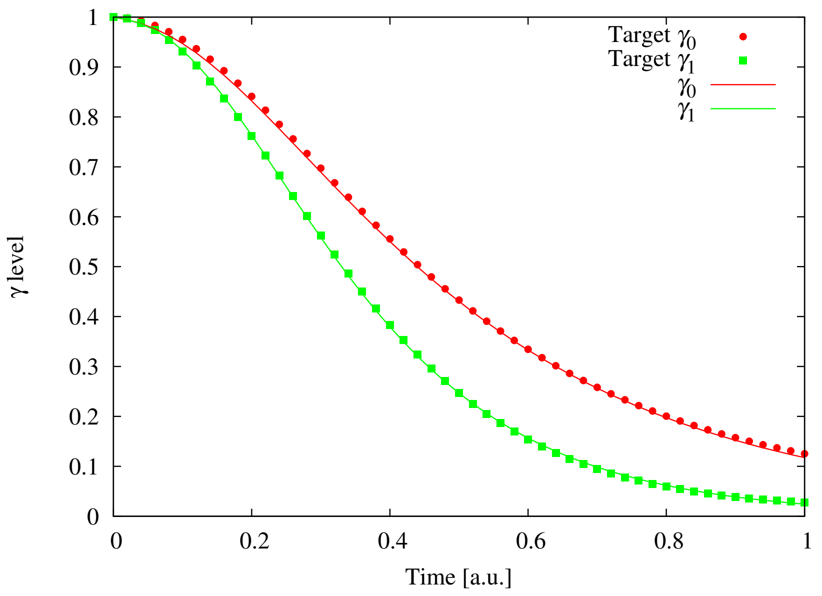


Figure 8.4: Comparison of the target dynamics (dots) with the simulated dynamics of the GRM inferred by cuGENED (lines). The GRM $\eta_{2\gamma}$ achieves a perfect fitting of the desired behavior.

All the reactions in $\eta_{2\gamma}$ are biologically consistent and well contribute to the definition of a network of plausible genetic interactions, which could be implemented by means of Synthetic Biology techniques. For instance, in $\eta_{2\gamma}$ we can observe the presence of a multi-product reaction (R_6) [80] and of alternative splicing reactions (R_4 and R_5) [203]. Furthermore, $\eta_{2\gamma}$ is characterized by a low fitness value, which indicates a consistent similarity with the expected target dynamics obtained by using the kinetic constants $(k_1, \ldots, k_8)$ inferred by PSO (Figure 8.4). Finally, it is worth noting that this optimal solution does not exploit all the chemical species in the set Σ: the CGP avoided the bloating and evolved a well fitting GRM that only contains 4 out of the 8 possible species in Σ.

8.3.2 ED of synthetic circuits with three genes

As a further test, the ED was performed on a system consisting of three genes, using the best settings of cuGENED identified during the preliminary tests (Section 8.3.1). For the ED of this synthetic circuit the settings were $|\Gamma| = 3$ and $|\Sigma| = 12$, so that

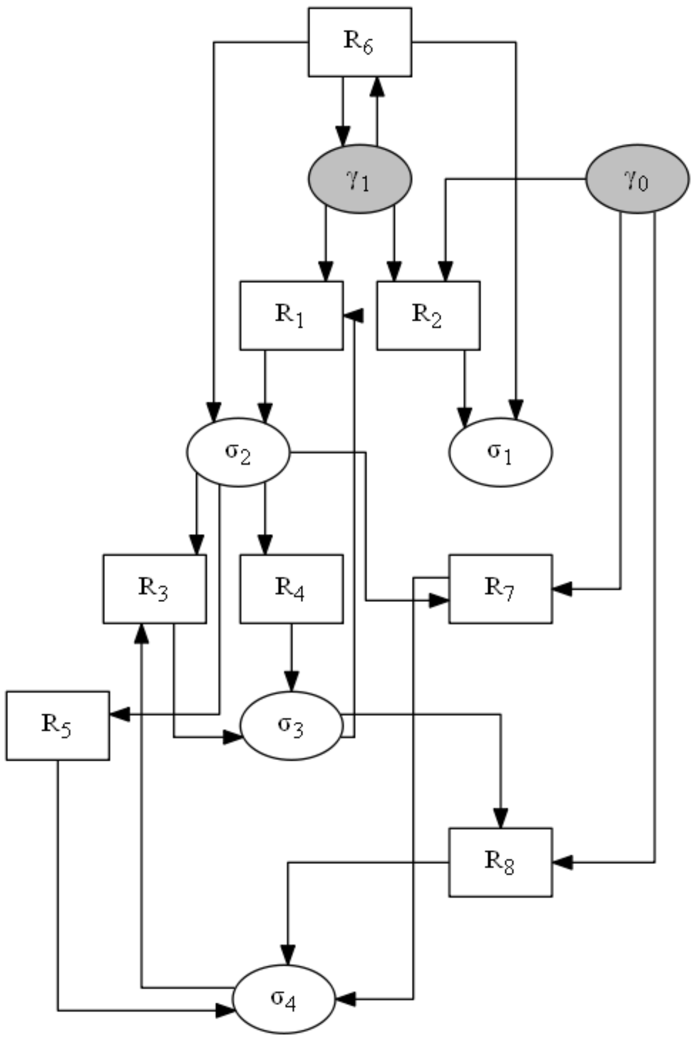


Figure 8.5: The interaction diagram shows the best GRM $\eta_{2\gamma}$ inferred by cuGENED to reproduce the desired behavior shown in Figure 8.4. Circular nodes represent the chemical species involved in the network, while rectangular nodes represent the reactions. The gray nodes denote the chemical species (i.e., mRNA) whose dynamics is considered as target.

the inferred GRM can contain at most $n_i = 15$ chemical species and $n_o = 51$ reactions (according to the heuristics described in Section 8.2).

The desired target dynamics of the species in Γ are described by two sigmoidal curves, characterized by different slopes, and a monotonically increasing curve (Figure 8.6, dots). This behavior corresponds to a down-regulation of two genes and an up-regulation of a third gene, whose underlying regulatory interplay is to be inferred by means of cuGENED.

The best GRM $\eta_{3\gamma}$ inferred by cuGENED is represented in Figure 8.7 and consists in the following reactions:

$$
\begin{aligned}
R_1 &: \gamma_0 + \sigma_4 \xrightarrow{4.679} \sigma_3,\\
R_2 &: \gamma_2 + \sigma_{10} \xrightarrow{20.491} \gamma_2,\\
R_3 &: \gamma_1 + \sigma_4 \xrightarrow{8.416} \sigma_3 + \sigma_{10},\\
R_4 &: \sigma_{10} \xrightarrow{4.654} \sigma_0,\\
R_5 &: \gamma_1 + \sigma_2 \xrightarrow{21.831} \gamma_1 + \sigma_4,\\
R_6 &: \sigma_{10} \xrightarrow{10.321} \sigma_{11},\\
R_7 &: \sigma_6 + \sigma_{10} \xrightarrow{14.681} \sigma_{11},
\end{aligned}
$$

where the numerical values above the arrows correspond to the best kinetic parameterization determined by PSO.

The result in Figure 8.6 shows that the simulation of $\eta_{3\gamma}$ almost perfectly fits the desired dynamics. In particular, the target and simulated dynamics for γ_0 and γ_1 are perfectly overlapped, while the curve of γ_2 slightly diverges from the expected behavior. Nevertheless, the simulated behavior of γ_2 is qualitatively and quantitatively similar to the target behavior, which means that, from a biological standpoint, the ED goal is fully achieved.

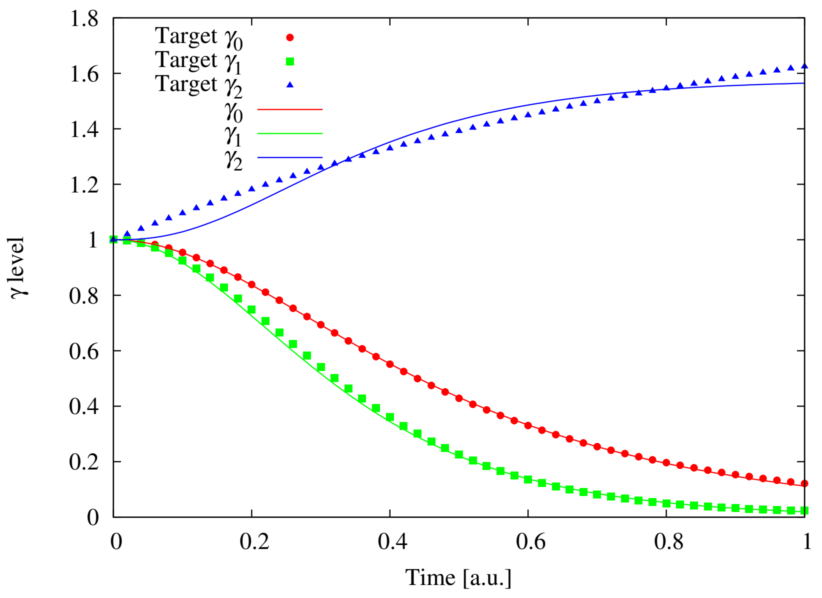


Figure 8.6: Comparison of the target dynamics (dots) with the simulated dynamics of the GRM inferred by cuGENED (lines). The GRM $\eta_{3\gamma}$ achieves an almost perfect fitting of the desired behavior.

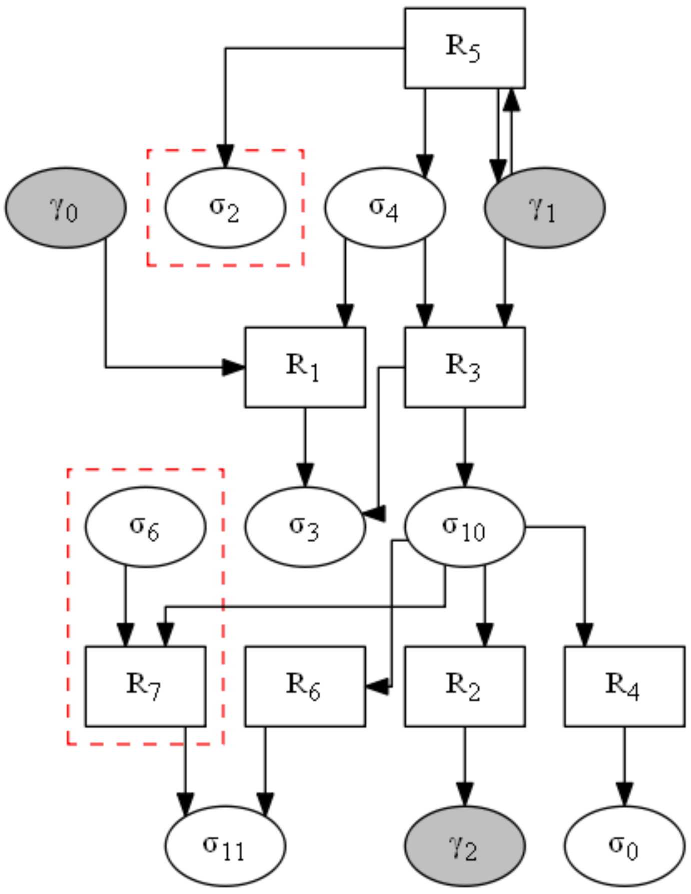


Figure 8.7: The interaction diagram shows the best GRM $\eta_{3\gamma}$ inferred by cuGENED to reproduce the desired behavior shown in Figure 8.6. Circular nodes represent the chemical species involved in the network, while rectangular nodes represent the reactions. The gray nodes denote the chemical species (i.e., mRNA) whose dynamics is considered as target for the ED. Dashed lines highlight reactions and chemical species that have no effective role in the system dynamics.

The interaction network in Figure 8.7 has two important characteristics, highlighted by the red dashed lines, which represent side effects of cuGENED:

- the chemical species σ_2, which is a product of reaction R_5, is not involved in the rest of the network. Hence, this species can be removed from $\eta_{3\gamma}$ without any impact on the system dynamics. It is worth noting that the role of the chemical species σ_3 is different from σ_2, because it is fundamental for the degradation of γ_0 as a consequence of reaction R_1;
- the reactants of reaction R_7 are species σ_6 and σ_{10}: whilst the latter is produced by reaction R_3, the former is not produced by any other reaction, so that the

level of σ_6 remains fixed to 0 during the simulations. The consequence is that reaction R_7 will never take place and hence it can be removed from $\eta_{3\gamma}$, along with σ_6. On the contrary, species σ_{11}, the product of reaction R_7, cannot be removed from the network, being involved also in reaction R_6.

8.3.3 Computational results

Using a GPU Nvidia GeForce GTX 590 equipped with 1024 cores, the ED of $\eta_{2\gamma}$ required 70136 sec (about 19 hours), while $\eta_{3\gamma}$ required 72645 sec (about 20 hours). Thus, the running time of cuGENED presumably scales less than linearly with the number of genes, making it feasible for the ED of systems characterized by a large number of genes. The increased running time is probably due to the different size of the candidate networks: the GRMs for the three-genes system are indeed characterized by a larger number of species and reactions, a circumstance that inevitably slows down the simulations. However, thanks to the use of the cupSODA deterministic simulator [230], the fitness evaluations are computed in parallel, so that the running time is not affected by the number of particles used in the PSO phase.

In order to compare the performances of cuGENED using CPU and GPU implementations of LSODA, 640 simulations are run — corresponding to 64 random parameterizations of 10 random GRMs, each one determined by a CP — which are equivalent to a single CGP iteration. As a reference CPU implementation it was used Scipy's `odeint` integrator [148], based on LSODA from the FORTRAN library odepack. The running time to perform the simulations was 1.5 sec on the CPU, and 0.061 sec on the GPU by using cupSODA, corresponding to a 24.5× speedup achieved by cuGENED.

8.4 Discussion

cuGENED consists in the integration of two evolutionary algorithms which work at two different levels: CGP to infer the structure of a GRN, and PSO to estimate the kinetic parameters of the reactions involved in the network. To the best of my knowledge, cuGENED represents the first attempt to tackle the ED of GRMs by means of CGP combined with PSO. This methodology is also the first attempt to automatically model a GRN assuming 0-knowledge on the set of molecular species. In this context, there are some issues that are worth considering.

In cuGENED, the ED problem is formalized without any *a priori* information on the structure of the system, defined in terms of molecular interactions, except for a desired target dynamics of a small subset of species. On the one hand, this information is sufficient to properly reconstruct a network whose dynamics fits the target data (see, for instance, Figure 8.4). On the other hand, this information alone is usually not enough to discriminate between possibly different network topologies [63], which could all be able to reproduce the same behavior used as target of the optimization. This well known issue in ED and RE is named *indistinguishability* of equivalent networks [327] (see also Section 10.3). To lessen this computational difficulty, a good practice consists in the multiple execution of the ED process, followed by an analysis of the obtained solutions to assess their laboratory feasibility. This analysis could as well be performed by means of automatic algorithms, which exploit a knowledge base of feasible reactions specifically defined by synthetic biologists.

With respect to other existing methodologies for the inference of GRNs, an advantage of cuGENED is that, thanks to the use of CGP, it allows to exploit a simple representation of the candidate GRMs. The genotype of CGP, indeed, is a fixed-length vector of integer values, whose corresponding phenotype is a set of human-readable chemical reactions. Moreover, cuGENED requires the estimation of only M kinetic parameters associated to the reactions where, in general, $M \ll (N^2 + 1)/2$. On the contrary, other existing GP methodologies generally necessitate complex parameterizations. For instance, S-tree based methods require the estimation of $2N(N + 1)$ kinetic parameters [57], thus making this method hardly exploitable for systems involving many genes. Moreover, thanks to the explicit limitation to second-order reactions in GRMs, cuGENED does not generate families of candidate solutions having an identical structure but a different stoichiometry, which still represents an open issue in GP-based methodologies [197].

Another relevant strength of cuGENED is the ED of synthetic circuits formally defined as GRMs which, in contrast to most existing approaches [44], are not based on arbitrary kinetics functions. The GRNs evolved by CGP and PSO consist in mass-action reaction-based models, which describe the biological processes in terms of simple molecular interactions, and not as reaction rate approximations based on some kind of chemical assumption. As a matter of fact, Hill functions were shown not to represent physically realistic reaction schemes [351], especially for gene expression processes [295], while the equilibrium or quasi steady-state assumptions at the basis of Michaelis-Menten constants are valid only in specific conditions [226].

A current drawback of cuGENED is that the choice of its initial setting is relevant to converge to an optimal solution, since both CGP and PSO are not settings-free algorithms. Common literature settings were exploited for the PSO, whilst the impact of different parameters values on the performances of CGP (i.e., the mutation rate ρ and the number of rows n_r) were analyzed. As a future improvement of cuGENED, it will be investigated the relationship between these two values, the number of genes and the complexity of the desired target dynamics, in order to derive a heuristic for the *automatic* selection of the most appropriate settings.

cuGENED relies on the nested execution of two evolutionary algorithms. As a consequence, its overall computational cost can be relevant, being proportional to the number of generations of CGP, to the number of iterations of PSO, to their population size, to the number of reactions involved in the candidate networks, and to the time length of the simulations. The largest part of the running time is due to the simulation of the dynamics of the candidate solutions, which is a fundamental step for the fitness computation. However, the computational cost can be strongly reduced by exploiting the GPU-powered cupSODA simulator [233]. So doing, it was achieved a parallel execution of the simulations and the fitness calculations of all candidate solutions, achieving a 24.5$\times$ speedup with respect to a strictly sequential execution of the same tasks. It is worth noting that the computational time could be further reduced by executing parallel ED instances. To the best of my knowledge, cuGENED also represents the first attempt ever in exploiting GPUs to perform an accelerated ED of GRMs.

8.5 Future perspectives

The idea of a circuit-like connectivity between biological parts was postulated for the first time in the 60s [217]. This intuition lead to several attempts to properly formalize biological regulation systems through mathematical models [106, 107, 155, 296] and to analyze the cellular pathways under investigation by exploiting electrical circuit analogies [208, 209]. This research field was favored by the progression of molecular biology and genomics, which provided several methods and the necessary knowledge to physically assemble biomolecular components. As a matter of fact, it is nowadays possible to properly evaluate the interaction among different genes through expression data coming from high-throughput techniques like microarrays [98] and real-time PCR [170]. It is also possible to engineer *in vitro* or *in vivo* customized signaling circuits [186]. The proof of concept that a computing-like behavior could be applied to biological systems was the design of the first synthetic gene networks, realized by using engineering-based methodologies [84, 94]. Lately, several network models were proposed, all of them integrating biochemical pathway information and expression data [6, 33, 195, 254].

In this context, cuGENED was developed: a computational methodology for the automatic design of GRNs characterized by a predefined dynamical behavior. In particular, cuGENED allows the derivation of mechanistic models of gene regulation system, modeled as a parameterized set of biochemical reactions. The reactions describe the processes related to gene expression, and they involve a set of molecular species (e.g., genes, mRNA) whose dynamics represent the target behavior of the optimization.

It is often the case that a gene can regulate either itself (autoregulation) or a single other gene. This is due to the fact that a generic regulatory species might be able to interact with a single component of the gene network through a *unique* DNA sequence. This means that all species could be extremely selective for their molecular target (i.e., each species can bind to a unique DNA sequence), so that no unknown interactions can occur between the regulators and their corresponding molecular targets [93]. Therefore, the molecular target of a regulator could be either the gene itself or another gene, for example in a very selective feedback system. Thus, as future developments of this work, the plan is to improve the ED methodology as follows:

- cuGENED will include the verification of the requirements about selective regulation to narrow the search space of candidate solutions, possibly obtaining GRMs that could be easier to implement with laboratory techniques;
- cuGENED will be modified to exploit the massive parallelism provided by GPUs to perform many parallel ED processes, in order to collect a set of optimal GRMs. This set can then be analyzed to identify the GRMs that show a better selective regulation (i.e., the GRMs that better fulfill the requirements stated above).

cuGENEND will also include an automatic mechanism to detect the unessential species (e.g., products that have no impact on the system dynamics), as well as the reactions that cannot take place (i.e., reactions whose reactants have zero concentration, like those highlighted in Figure 8.7). In addition, the future implementation will consider the fact that the inference process might converge to a GRM whose simulation perfectly fits the desired behavior, though some reactions appearing in the GRM might not be biologically plausible. As mentioned in Section 8.4, a solution to this problem might come from the inclusion of some knowledge domain constraints in the ED, in order to automatically remove such reactions from the candidate GRMs.

Finally, in cuGENED the biological noise that is typical of gene expression [82] is not considered. Actually, some species involved in GRNs can have very low intracellular amounts (in the order of tens or a few hundreds of molecules), so that a stochastic simulation methodology could be preferable with respect to deterministic simulations to correctly reproduce any noise-induced emergent phenomena. cuGENED can be straightforwardly extended to support also stochastic simulation algorithms for the simulation of the temporal dynamics of candidate solutions [105, 235, 291]. Indeed, as a further development, it will be investigated the possibility of the automatic reconstruction of networks in presence of intrinsic noise, oscillations or multi-stability phenomena.

Chapter 9

Protein structure inference

The determination of molecular structures is of great interest both in chemistry and biology, since the three-dimensional (3D) shape of a molecule is the main determinant of its function. In many contexts, such as drug discovery, metabolic engineering and catalysis, structural information is essential to understand and control the behavior of a molecular system. The great majority of structural data available today arise from two experimental techniques: X-ray crystallography and Nuclear Magnetic Resonance (NMR) [360].

NMR exploits the magnetic properties of the nucleus of isotopes (as ^{1}H, ^{13}C and ^{31}P) to identify spatial neighborhood relationships between chemical groups, which are generally given in the form of a matrix of inter-atomic distances. When this technique is applied to molecules of significant size and with a complex 3D shape, the resulting distance matrix is both sparse and noisy due to technical limitations of NMR.

This is exactly the case of proteins, an ubiquitous class of biological molecules characterized by a great variability in shape and size. The *Molecular Distance Geometry Problem* (MDGP) consists in reconstructing the 3D structure of a molecule starting from its (sparse) distance matrix. The MDGP problem is a special case of the Distance Geometry Problem (DGP) [114] in which the distance matrix is obtained from NMR experiments. The peculiarity of the MDGP relies on the availability of additional constraints on the inter-atomic distances in the 3D structure, which can be defined according to the chemical and physical properties of the class of molecules under investigation.

In the case of incomplete information in the distance matrix, the MDGP was shown to be NP-hard [299] by reducing a 1-dimensional MDGP to the SUBSETSUM problem [219]. Several different approaches to solve the MDGP have been proposed in recent years, but they all suffer from limitations (see Section 9.1). For instance, the geometric buildup [76] is unable to find a solution to the problem for some cases of sparse distance matrices; the branch and prune algorithm [173, 174] has an exponential complexity; ABBIE [129], EMBED [65] and DGSOL [220] algorithms allow to obtain only approximate solutions to the MDGP.

In order to overcome some limitations of the existing methodologies for the MDGP, and considering that some problems of the NP class can be efficiently tackled by means of soft-computing and population-based algorithms, this chapter proposes a memetic algorithm (MA, see Section 3.4 for details) combining Swarm Intelligence (Section 3.3) and Evolutionary Computation (Section 3.2), together with a local search algorithm (GD, Section 3). In particular, the idea is to combine the swarm-based optimization of PSO with the crossover capabilities typical of GAs. SI is used to move atoms belonging to a candidate solution (i.e., a 3D molecular structure) within the search space. Each solution is characterized by a different position of atoms, whereby even solutions with the same fitness value can have atoms with a completely different position due to roto-translations of the whole structure. A crossover operator, typically employed in GAs, is used to exchange substructures between candidate solutions; in this context, a local optimization method is exploited to find the optimal roto-translation of the exchanged substructure within the offspring solution.

Besides the inter-atomic distance matrix, the MA makes use of additional constraints:

1. the size of the search space where particles move is bounded according to the number of amino-acids of the proteins;

2. molecular chirality — a property of asymmetry that is imposed to protein structures — is considered during the optimization process.

The main contents of this chapter are published in the proceedings of the 2014 IEEE Congress on Evolutionary Computation [237].

9.1 The Molecular Distance Geometry Problem

The MDGP can be formulated as follows. Let $\mathfrak{N}$ be the number of atoms in a protein π, and let $d_{ij} \in \mathbb{R}^+$ be the given distance between atoms i and j, with $i, j = 1, \ldots, \mathfrak{N}$ and $i \neq j$, measured according to some computational or experimental methodology (e.g., NMR). These distances can be arranged into a real-valued $\mathfrak{N} \times \mathfrak{N}$ matrix $\mathbf{d}$, such that d_{ij} is the value in the i-th row and j-th column in $\mathbf{d}$.

If we denote by $\mathbf{a}_i$ the 3D coordinate vector of atom i in the Euclidean space, i.e., $\mathbf{a}_i = (x_i, y_i, z_i) \in \mathbb{R}^3$ for each $i = 1, \ldots, \mathfrak{N}$, then the value d_{ij} will formally correspond to the Euclidean norm between the two atoms, i.e., $d_{ij} = \|\mathbf{a}_i - \mathbf{a}_j\| = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$; note that, anyway, these coordinate vectors are unknown.

The MDGP consists in finding the set of coordinate vectors $\mathbf{a}_1, \ldots, \mathbf{a}_{\mathfrak{N}}$ of all atoms in π, such that the Euclidean norm $D_{ij} = \|\mathbf{a}_i - \mathbf{a}_j\|$ between any pair of atoms i and j — evaluated according to these coordinates — is equal to the measured distance d_{ij}. Formally, the MDGP is solved if $D_{ij} = d_{ij}$ for all $i, j = 1, \ldots, \mathfrak{N}$, $i \neq j$.

Several approaches to MDGP have been proposed in recent years. Dong and Wu introduced a linear time algorithm, called "geometric buildup", to solve the 3D-DGP when the exact value of distances between all pairs of atoms are given [76]; recently, this approach has been extended in order to obtain an approximate solution for the MDGP with noisy distance values and sparse matrices [314, 315]. The main limitation of the geometric buildup strategy is that in the case of sparse matrices and, in particular, when some atoms are characterized by less than four distance constraints, this method is unable to find any solution. However, to overcome this limitation, it is possible to consider additional distance constraints arising from structural features of proteins, or using optimization algorithms [88], to reconstruct the complete molecular structure from a partial substructure obtained with the geometric buildup algorithm.

A branch and prune algorithm was proposed in [173, 174]: by exploiting additional constraints about the protein structures, this method considers a discrete search space in which the amino-acids can be placed only in two different positions with respect to their precursor in the protein structure. This algorithm has an exponential complexity, however it is able to efficiently find solutions for some instances that satisfy particular structural properties.

There exist two approaches based on graph embedding [277] in 3D Euclidean space, able to deal with both noisy data and sparse matrices. The first one, called ABBIE algorithm [129], exploits a divide and conquer strategy and structural rigidity [286]; this method first identifies subproblems (i.e., subsets of nodes) that can be solved with an exact algorithm, then it applies a global optimization algorithm to combine partial solutions. The second one, called EMBED algorithm [65], uses the measured distances to derive a set of lower and upper bounds for all other distances; this requires the identification of the shortest path between each couple of nodes in a particular graph in order to derive triangle inequality limits [126]. A local optimization strategy is then applied to refine the solution obtained from the complete bounds set.

Finally, the DGSOL algorithm [220] combines a methodology to select good starting points for the optimization process with the Gaussian smoothing and continuation strategy [258], a technique used to reshape the objective function. So doing, a gradient minimization can be applied to the obtained smooth function in order to optimize the protein structure. The main limitation of DGSOL is that it provides only approximate solutions in presence of noisy information and sparse matrices.

9.2 Structure inference using a hybrid memetic algorithm

A candidate solution of the MDGP can be encoded as a vector $\mathbf{\Pi} = (\mathbf{a}_1, \dots, \mathbf{a}_{\mathfrak{N}})$ of 3D coordinates, representing the positions of all atoms of protein π in the Euclidean space. This representation can be exploited by a traditional evolutionary methodology as GAs, with genetic operators specifically designed to work on candidate solutions encoding real values (see Section 3.2.1 for further details on real-coded GAs). Even though GAs might be a feasible methodology for MDGP, SI techniques like PSO are generally more suitable than GAs, since they natively optimize real-valued problems [157]. Nevertheless, the crossover operator of GAs — which exchanges the genetic material of two promising individuals to create an improved offspring generation — is an elegant and powerful means to obtain a recombination of individuals and a better exploration of the search space. Thus, in this chapter, a hybrid methodology combining swarm-based optimization of PSO with crossover capabilities of GAs is proposed. Crossover, in this case, implements the exchange of a subset of atoms between individuals, i.e., substructures of the candidate solutions. A substructure is defined as $\sigma = (i_1, \dots, i_{\mathcal{K}})$, $\mathcal{K} \leq \mathfrak{N}$, i.e., the vector of indexes corresponding to a subset of atoms positions $(\mathbf{a}_{i_1}, \dots, \mathbf{a}_{i_{\mathcal{K}}})$ of solution $\mathbf{\Pi}$.

During the PSO optimization phase, the atoms belonging to each candidate solution move inside the search space and can be placed in completely different positions, so that even if two individuals are characterized by the same fitness value they might be rotated or translated with respect to each other in the 3D space. As a consequence, the crossover operator might move a substructure from an individual into another in such a position that the new (offspring) molecule will have a worse fitness value, because of an uncontrolled scattering of atoms. To reduce this potentially deleterious impact of crossover, a local optimization by means of a GD algorithm (see Section 3)is also performed to optimize the roto-translation that must be applied to the substructure. As described in Section 3.4, global optimization methods coupled to local search are called MAs [169], thus the methodology described in this Chapter is defined as a Memetic Hybrid PSO plus GAs (MemHPG).

9.2.1 A Memetic Hybrid Methodology for MDGP: MemHPG

The MemHPG algorithm combines the emergent, self-organizing behavior of swarms with the strength of crossover-based recombination. Self-organization is performed by a modified version of PSO which is used to arrange the atoms in the search space of a candidate structure; the crossover, on the other side, is applied to a population of independent candidate structures, to exchange their optimal substructures. As a matter of fact, the hybrid algorithm works on two layers: the inner layer of atoms (hereby called the *PSO-layer*) and the outer layer of molecule structures (the *GA-layer*). Figure 9.1 reports a schematization of the two layers.

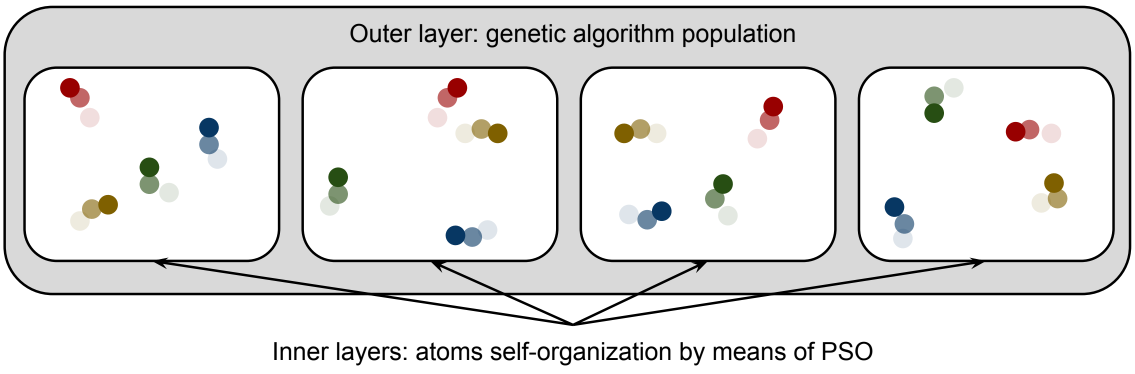

Outer layer: genetic algorithm population
Inner layers: atoms self-organization by means of PSO

Figure 9.1: Schematization of the two-layer hybrid methodology. In the outer layer, a population of candidate solutions exchange promising substructures exploiting GAs crossover, while each candidate solution evolves in the inner layer by means of PSO.

PSO-layer. In this modified version of PSO, a single particle for each atom is used, so that the position $\mathbf{x}_i$ of the i-th particle here corresponds to the 3D Euclidean coordinate vector $\mathbf{a}_i$ of the i-th atom of protein π. Therefore, in this particular formulation, particles do not represent a solution to the MDGP problem; instead, they represent a solution for the sub-problem of identifying the optimal spatial positioning of atoms. The size of the search space for particle positioning was defined according to the number A of amino-acids in protein π (which is known *a priori*), considering the notion of radius of gyration of proteins [139] (whose upper bound was identified as $A^{3/5}$). In MemHPG, the best setting for the search space was empirically found to be $4 \cdot A^{3/5}$Å for each dimension in the 3D Euclidean space.

The initial position of particles is randomly generated within the search space, except for the particle corresponding to the first atom in π which is placed in position $(0, 0, 0)$ and kept fixed during the optimization. The rationale behind this choice is that, by keeping one particle fixed, the rest of the swarm is constrained to self-organize around it, thus reducing the chaoticity of the overall movements.

Once particles are distributed in the search space, an *error* ε is calculated to estimate the precision of the corresponding candidate solution, considering only the given distance constraints d_{ij}, without any additional knowledge about the original structure:

$$\varepsilon = \frac{\sum_{i=1}^{\mathfrak{N}} \sum_{j=1}^{\mathfrak{N}} \eth_{ij}}{\sum_{i=1}^{\mathfrak{N}} \mathcal{G}_i}, \tag{9.1}$$

where

$$\eth_{ij} = \begin{cases} 0 & \text{if } d_{ij} \text{ is not given} \\ |D_{ij} - d_{ij}| & \text{otherwise,} \end{cases} \tag{9.2}$$

and the value $\mathcal{G}_i$ denotes the number of atoms $j \neq i$ in π for which a distance value d_{ij} is given. Since Equation 9.1 allows to discriminate the quality of solutions, it is exploited as the fitness function.

At each step of the PSO procedure in MemHPG, each particle considers a novel kind of attractor, named the *aggregate attractor* and denoted by $\mathbf{h}_i(t) \in \mathbb{R}^3$, which is calculated by comparing the distance between the coordinate vectors of all other atoms in the candidate solution (D_{ij}) against the distance measured with NMR experiments (d_{ij}):

$$\mathbf{h}_i(t) = \sum_{j \neq i} \frac{\delta_{ij}}{D_{ij}} (\mathbf{a}_j(t) - \mathbf{a}_i(t)), \tag{9.3}$$

where $\delta_{ij} = D_{ij} - d_{ij}$ is used to weight the attraction between atoms, so that the contribution to the aggregate attractor of atoms whose distance D_{ij} is close to the measured distance d_{ij} will be reduced. It is worth noting that the aggregate attractor $\mathbf{h}_i$ can be seen as a linear combination of $\mathfrak{N} - 1$ "global" attractors of particle $\mathbf{a}_i$, each one with a different social factor equal to δ_{ij}.

When two atoms are farther than expected, they act as mutual attractors; on the contrary, when the atoms are closer than expected, they behave as repulsers. Figure 9.2 provides two examples of this mechanism, represented in the x-y projection plane for the sake of simplicity. According to Equation 9.3, the aggregate attractor for atom i is calculated as the sum of all attractive/repulsive contributions of all other atoms $j \neq i$; Figure 9.3 shows an example which considers the aggregation of the contributes due to two atoms.

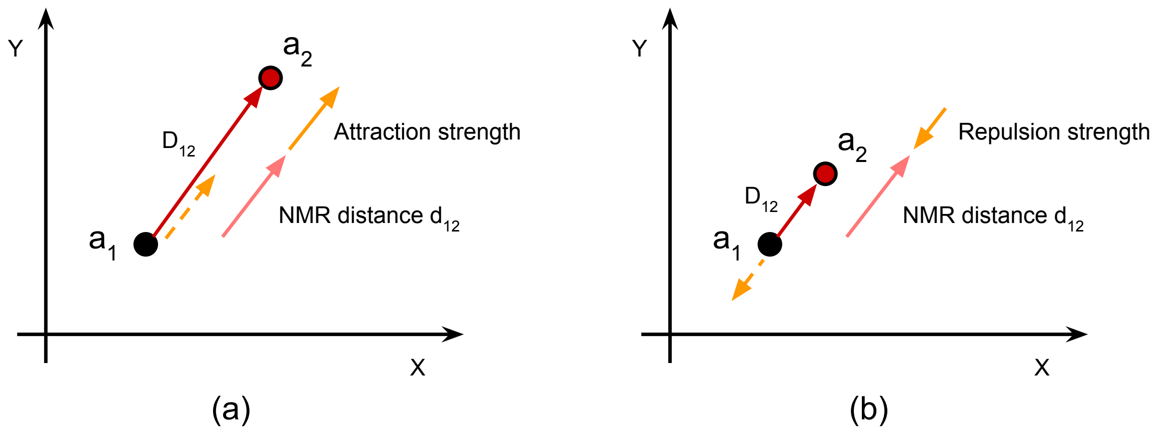


Figure 9.2: Example of the attraction/repulsion mechanism of the modified PSO. For the sake of clarity, only the vectors for particle $\mathbf{a}_1$ are shown. (a) When the distance between two atoms (the red arrow between $\mathbf{a}_1$ and $\mathbf{a}_2$) is larger than the one measured by NMR (pink arrow), the atoms attract each other (dashed yellow arrow). (b) When the distance between the two atoms is smaller than the distance measured by NMR, the atoms act as repulsers.

In order to consider only this new attractor in the modified version of PSO, Equation 3.4 is modified as follows:

$$\mathbf{v}_i(t+1) = w \cdot \mathbf{v}_i(t) + \mathbf{r} \circ \mathbf{h}_i(t), \tag{9.4}$$

where $\mathbf{r}$ is a vector of random numbers uniformly sampled in [0,1].

Once the putative velocity $\mathbf{v}_i(t+1)$ is calculated, its velocity is clamped, i.e., if $||\mathbf{v}_i(t+1)|| > v_{\texttt{MAX}}$ then

$$\mathbf{v}_i(t+1) = \frac{\mathbf{v}_i(t+1)}{||\mathbf{v}_i(t+1)||} \cdot v_{\texttt{MAX}}$$

and the position $\mathbf{a}_i(t+1)$ is updated according to Equation 3.5. During the last generations of MemHPG, the finer positioning of atoms in the candidate structures requires smaller and more controlled movements with respect to the initial phases. For this reason, the methodology self-adapts the $v_{\texttt{MAX}}(t)$ value as follows:

$$v_{\texttt{MAX}}(t) = \begin{cases} \alpha \cdot v_{\texttt{MAX}}(t-1) & \text{if } \varepsilon^*(t) > \varepsilon^*(t-1) \\ v_{\texttt{MAX}}(t-1) & \text{otherwise,} \end{cases}$$

where $\varepsilon^*(t) \in \mathbb{R}^+$ represents the smallest error value among all particles at generation t and $\alpha \in (0,1)$ is the velocity adaptation factor. The iterative update of velocity vectors, calculated according to the aggregate attractor, allows the set of atoms to self-organize in a single optimal position. The inertia weight and the randomness due to $\mathbf{r}$ allow particles to avoid a chaotic behavior and local optima.

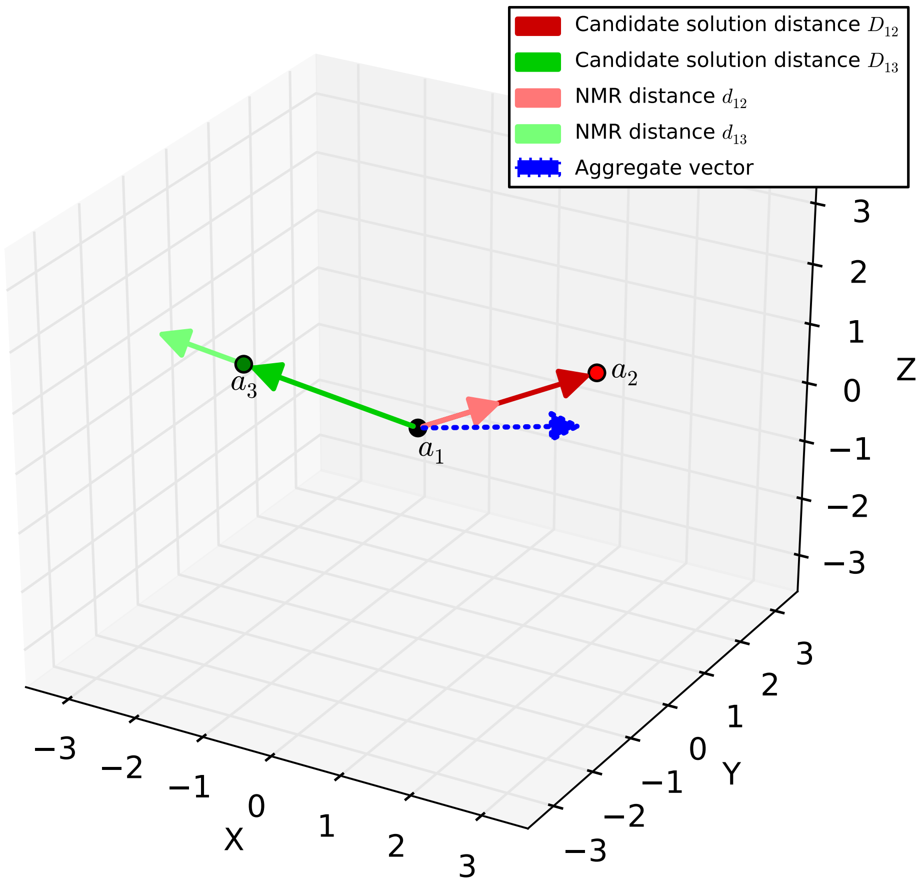


Figure 9.3: Example of calculation of the aggregate attractor for particle $\mathbf{a}_1$, in a 3-atoms system. The length of the red arrows represents the distance between particles $\mathbf{a}_1$ and $\mathbf{a}_2$ according to the candidate solution (dark red) and to NMR data (light red): since the latter is shorter than the former, $\mathbf{a}_2$ acts as an attractor for $\mathbf{a}_1$. The length of the green arrows represents the distance between particles $\mathbf{a}_1$ and $\mathbf{a}_3$ according to the candidate solution (dark green) and to NMR data (light green): since the latter is longer than the former, $\mathbf{a}_3$ acts as a repulser for $\mathbf{a}_1$. The resulting aggregate attractor $\mathbf{h}_1$ is represented by the blue vector. The same process is applied to particles $\mathbf{a}_2$ and $\mathbf{a}_3$ (not shown here).

GA-layer. To help the convergence to an optimal solution, MemHPG introduces a second layer by instantiating Q multiple independent candidate solutions, which constitute the population $\mathcal{P}$ of a GA. MemHPG does not exploit a mutation operator, which is conceptually realized by PSO: the GA performs tournament selection and crossover only. These operators are applied every I_χ iterations, and work together to generate the offspring population.

The functioning of the GA-layer is summarized in the following steps:

- a subset $\mathcal{P}_{\texttt{TOUR}} \subset \mathcal{P}$ of q individuals, $1 < q < Q$, is sampled using a uniform distribution;

- the best individual $P_{\texttt{BEST}} \in \mathcal{P}_{\texttt{TOUR}}$ is deterministically identified (according to the fitness values) and copied into the new population $\mathcal{P}'$;
- for each atom i, such that $\mathbf{a}_i \in P_{\texttt{BEST}}$, a substructure σ_i is identified as explained below and inserted in a set $\mathfrak{S}$;
- one element $\sigma \in \mathfrak{S}$ — that is, a protein substructure — is chosen with a probability proportional to its length (i.e., the number of atoms in σ);
- one individual $P_{\texttt{RND}}$ is randomly chosen from $\mathcal{P}_{\texttt{TOUR}} \setminus P_{\texttt{BEST}}$ with a uniform probability;
- the atoms in the substructure σ are positioned into individual $P_{\texttt{RND}}$ according to the best roto-translation (as explained below), thus replacing the corresponding atoms and generating a new individual $P^{\sigma}_{\texttt{RND}}$;
- finally, $P^{\sigma}_{\texttt{RND}}$ is inserted into population $\mathcal{P}'$ and the velocities of its particles are set to zero.

This procedure is repeated until $|\mathcal{P}'| = Q$; then, $\mathcal{P}'$ replaces $\mathcal{P}$.

The substructures in $\mathfrak{S}$ are chosen as follows. For each atom i in $P_{\texttt{BEST}}$, a substructure σ_i is determined according to the following greedy algorithm:

- atom i is inserted in σ_i;
- find atom j, $j \neq i$, such that $|\delta_{ij}| = \min\{|\delta_{ik}| \mid k = 1, \ldots, \mathfrak{N}, k \neq i\}$. If $|\delta_{lj}| < \varphi_{\texttt{min}}$, for each $l \in \sigma_i$ and $l \neq j$, then add atom j to σ_i; otherwise stop, as the substructure cannot be extended.

The value $\varphi_{\texttt{min}}$ is defined as $\varphi_{\texttt{min}} = \min\{\varphi_i \mid i = 1, \ldots, \mathfrak{N}\}$, where $\varphi_i = \frac{1}{\mathfrak{N}} \sum_{j \neq i} |\delta_{ij}|$. The procedure is iterated until no more atoms can be inserted in σ_i or its length reaches a given value $size_{\texttt{MAX}}$, that corresponds to a fixed percentage of the total number of atoms in π.

Since the chosen substructure σ can be oriented and translated in space in any possible way, its positioning in $P_{\texttt{RND}}$ is optimized: the crossover embeds a local search optimization to identify the best roto-translation of σ with respect to its surrounding atoms in $P_{\texttt{RND}}$. More precisely, this is done by first calculating the centroid of σ and of the corresponding subset of atoms that will be replaced in $P_{\texttt{RND}}$, and then exploiting a GD method to identify an optimal translation vector $\boldsymbol{\Upsilon} = (t_x, t_y, t_z)$ and an optimal rotation vector $\boldsymbol{\theta} = (\theta_x, \theta_y, \theta_z)$ with respect to these centroids which minimize the impact to the aggregate attractor. After the crossover process, the PSO starts again.

In addition, every 50 iterations of MemHPG, the *chirality* of candidate solutions is verified, since the information contained in the distance matrix is not sufficient to discriminate between a correct reconstructed molecule and molecules with a different chirality. Chirality is a property related to a lack of symmetry that regards organic molecules such as amino-acids (see Figure 9.4); chirality is observed when a carbon atom is bound to four different chemical groups (these carbon atoms are called chiral carbons) [121]. For each protein substructure composed of one chiral carbon and the atoms bound to it, two different but isometric conformations are possible, named *enantiomers*; therefore, one must apply a geometric transformation to adjust the right positioning of atoms in order to impose the specific conformation of these groups, that is typical of protein molecules [343]. The subset of atoms that can lead to a wrong reconstruction can be identified *a priori* by analyzing carbon atoms and their bound chemical groups.

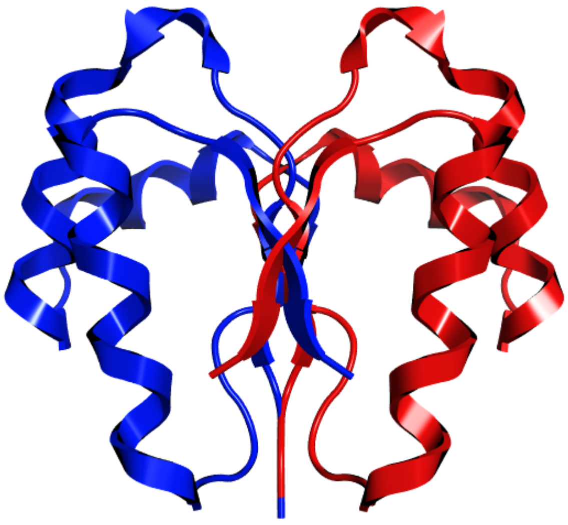


Figure 9.4: Example of two proteins characterized by the same atom-atom distances but different chirality. The two enantiomers (red and blue) have identical structure, except they are symmetrical and cannot be superimposed.

The procedure for chirality correction can be summarized as follows. For each candidate solution, the substructures whose chirality is not correct are identified and modified by means of geometric operations like rotations and reflections in the Euclidean space, according to the tetrahedral geometry of the chemical bonds of the chiral carbons. More specifically, the following steps are performed to identify and fix the chirality of a substructure:

1. the whole protein is roto-translated, in order to position the chiral carbon of the substructure in the origin of axis;

2. the protein is then rotated in order to align the hydrogen atom of the substructure to the x axis (with y and z components equal to 0);

3. at this point, it is possible to discriminate the chirality of the substructure by observing the position of the rest of the substructure, with respect to the y axis;

4. if the detected chirality is different from what is expected, the substructure is mirrored by changing the signs of the y components;

5. the whole protein is finally roto-translated in the original position, by applying the inverse transformation with respect to step 1.

This procedure is performed after each crossover process; when the chirality verification is completed, the PSO starts again.

MemHPG stops when a user-defined termination criterion is met, i.e., after a fixed number of iterations $IT_{\texttt{MAX}}$.

The computational complexity of MemHPG is relevant. For instance, the update of the positions of $\mathfrak{N}$ atoms for a population of Q individuals has a computational complexity of $O(Q \cdot \mathfrak{N}^2)$ per generation.

Each individual in the GA-layer is independent, so that the whole population can evolve in parallel. Moreover, for each individual, the atoms moving as a consequence of the PSO-layer are independent, so that this process can be parallelized as well. Thus, the complexity of the methodology can be strongly reduced by adopting a parallel architecture, as described in the next section.

9.2.2 GPU implementation

Among the available choices, the parallel version of method was implemented by using Nvidia's CUDA, to leverageGPU's horsepower (Section 4.1).

In order to have a logical mapping between MemHPG's entities and the CUDA execution hierarchy, a block was assigned to each GA individual; each thread of the block is responsible for the update of a specific atom of that individual, performing the calculations of the PSO-layer (see schematization in Figure 9.5). Following this execution model, a CUDA kernel was implemented to perform a single iteration of the evolutionary process, in which all atoms of all individuals are updated in parallel, reducing the computational complexity for the single protein update to $O(\mathfrak{N})$.

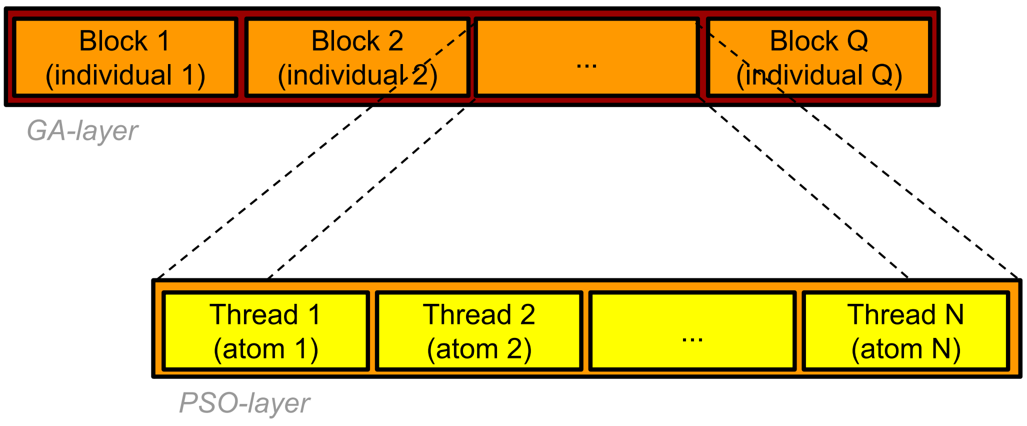


Figure 9.5: Schematization of MemHPG GPU implementation. Each individual of the GA-layer is assigned to a CUDA block; each atom/particle of the PSO-layer is assigned to a specific thread of the block.

A second kernel is responsible for the parallel calculation of the finess function (Equation 9.1), which is performed by means of a parallel reduction algorithm. This strategy reduces the computational complexity from $O(\mathfrak{N})$ down to $O(\log_2 \mathfrak{N})$. The computational effort is also reduced by exploiting the shared memory, which can be used to accumulate the partial results of the reduction algorithm, avoiding the high latencies due to the accesses to the global memory.

By iteratively repeating the kernels described above, the population of candidate proteins evolves. Moreover, the crossover operator is applied every I_χ generations. A parallel version of the operator is still under development, so that it is currently performed sequentially on the CPU, even though it exploits the partial calculations executed on the GPU during the atoms' update to accelerate the determination of the best substructures.

The implementation described above represents an elegant and performant alternative to a serial counterpart. Nevertheless, as described in Section 4.1, CUDA limits the number of threads in a block to 1024, so that MemHPG is currently limited to proteins characterized by at most 2^{10} atoms. This does not represent a relevant limitation, since NMR data can hardly produce meaningful distances of proitens larger than a thousand atoms. Nevertheless, an improved and "block unaligned" version of the algorithm, free from any protein size limitation, is currently under development. This topic is further discussed in Chapter 10.

9.2.3 Results

This section summarizes the results obtained by MemHPG for the reconstruction of the 3D structure of different proteins. As a first step, several tests were performed to determine the influence of the values of PSO and GA parameters on the reconstruction process, in order to find the best settings of MemHPG that were then exploited in all experiments. These tests consisted in the variation of a single parameter at a time in the optimization process of an *in silico* generated 3-peptide molecule with a length of $\mathfrak{N} = 56$ atoms. For each parameter, each test was repeated 30 times, and the average smallest error achieved with the different MemHPG parameterizations was used to evaluate the influence of that parameter. In all preliminary tests, $IT_{\texttt{MAX}} = 2000$ unless otherwise specified.

In the first test the impact of the population size Q was analyzed, by considering the following values: 32, 64, 128, 256 individuals. As expected, the average smallest error achieved decreases as the population size increases; however, for $Q > 32$, the improvement of the solutions quality is so slight that it does not justify the larger use of computational resources that it would require. Therefore, the value used in all consecutive tests was set to $Q = 32$.

In the second test, the impact of different values for the coefficient ϑ, used to clamp particles' initial maximum velocity, was analyzed. Multiple values for $v_{\texttt{MAX}} = D_{\texttt{MAX}}/\vartheta$, where $D_{\texttt{MAX}}$ is equal to the diagonal length of the search space, were tested. As shown in Figure 9.6, the best results were achieved when $\vartheta = 10$, while smaller values (e.g., $\vartheta = 1$) and higher values (e.g., $\vartheta > 500$) lead to worse results.

The third test consisted in varying the adaptive velocity factor α. Figure 9.7 shows the average smallest error obtained with 30 runs of MemHPG with several values of factor α. In this test, where $IT_{\texttt{MAX}} = 4000$, the best results were obtained with $\alpha = 0.999$, even if smaller values of this factor allowed a faster convergence.

A further test concerned the influence of the inertia weight on the particles velocity; in particular, the w value was varied in the range $[0, 1]$ and the best result was achieved with $w = 0.4$. Similarly to the case of $v_{\texttt{MAX}}$, both higher and smaller values of the inertia weight lead to higher values of the average smallest error.

The last three tests aimed at finding the best setting for the tournament size, the crossover frequency and the maximum length allowed for a substructure involved in the crossover operation. The best tournament size value was identified around 10% of the population size Q, in order to have a high selection pressure able to maintain the population diversity throughout the generations. The crossover frequency was set to $I_\chi = 50$, meaning that every 50 generations the individuals undergo this operator.

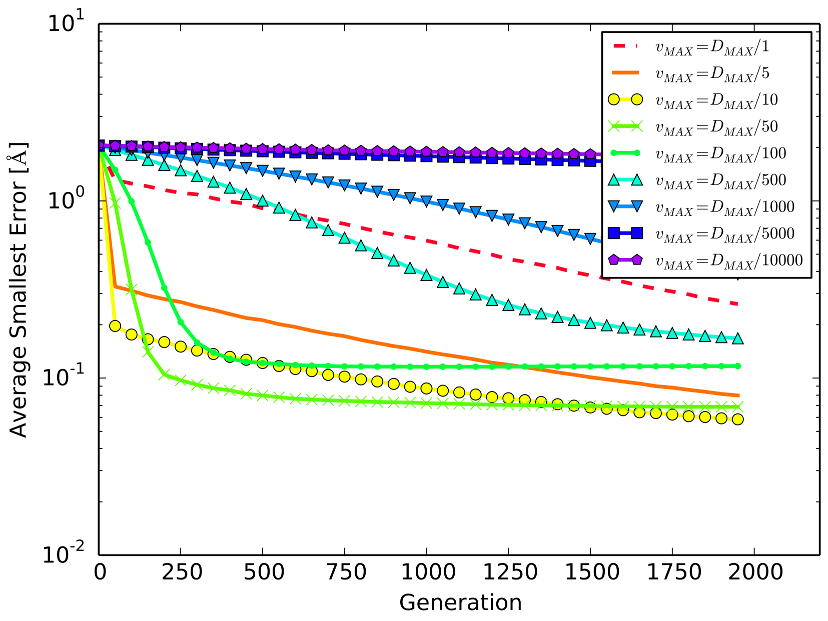


Figure 9.6: Average smallest error computed over 30 runs of MemHPG varying the coefficient ϑ in $v_{\texttt{MAX}} = D_{\texttt{MAX}}/\vartheta$. The best results were achieved with $\vartheta = 10$; note that both high and small values for the maximum velocity of particles lead to higher values of the average smallest error.

Despite the crossover improves the average quality of the candidate solutions, increasing its application frequency worsen the fitness of individuals. Finally, the maximum length allowed for a substructure involved in the crossover operation was set to $size_{\texttt{MAX}} = 15\%$ of the total number of atoms in π (for higher values better results can be achieved, but the improvement of the fitness is not enough to justify the larger use of computational resources that it would require).

The results of these preliminary tests led to the following best parameter settings for MemHPG:

- population size $Q = 32$ individuals;
- initial $v_{\texttt{MAX}} = D_{\texttt{MAX}}/10$;
- adaptive velocity factor $\alpha = 0.999$;
- inertia weight $w = 0.4$;
- tournament size $q = 4$ individuals;
- crossover interval $I_\chi = 50$ generations;

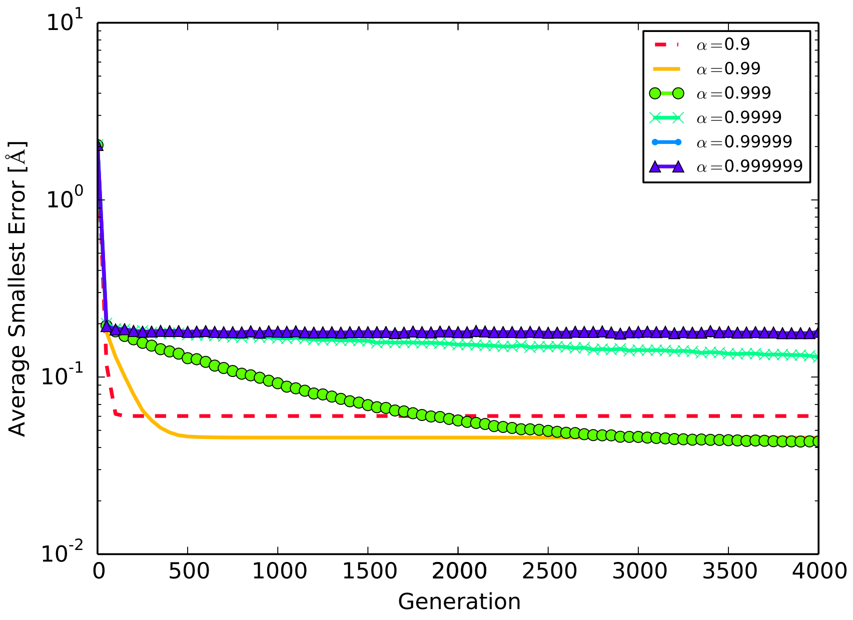


Figure 9.7: Average smallest error computed over 30 runs of MemHPG varying the adaptive velocity factor α. Even though for α equal to 0.9 or 0.99 a faster convergence was obtained, the value $\alpha = 0.999$ allowed to achieve the best results.

- $size_{\mathtt{MAX}} = 15\%$.

To test the validity of this setting of MemHPG, a 3-peptide molecule was reconstructed by using incomplete information. This was realized by removing from matrix $\mathbf{d}$ the distance values d_{ij} that are above a given cutoff. As shown in Figure 9.8, the average smallest error of the structures reconstructed by MemHPG is below 10^{-4}Å, also in the case of a matrix $\mathbf{d}$ where $d_{ij} < 6$Å for all $i, j = 1, \ldots, \mathfrak{N}$.

To show the effectiveness of the methodology, Table 9.1 shows the results obtained from the reconstruction of the structure of 9 proteins of increasing length — taken from the PDB database [88, 269] — using only inter-atomic distances $d_{ij} < 6$Å or $d_{ij} < 7$Å. In particular, for each protein, Table 9.1 reports the error ε (defined in Section 9.2.1) and the Root Mean Square Deviation (RMSD) [39] of the best structures found by MemHPG after $IT_{\mathtt{MAX}} = 20000$ iterations. These results highlight the robustness of the methodology since the ε value is low in all cases and, in addition, the RMSD is always lower than 3.5Å, a value that is considered to be indicative of a good reconstruction of protein structures [302].

In Figure 9.9, it is shown the structural alignment, realized with PyMOL [221], of the protein structures obtained with MemHPG (using inter-atomic distances below the 6Å cutoff) with the structures available in the PDB database. In the case of

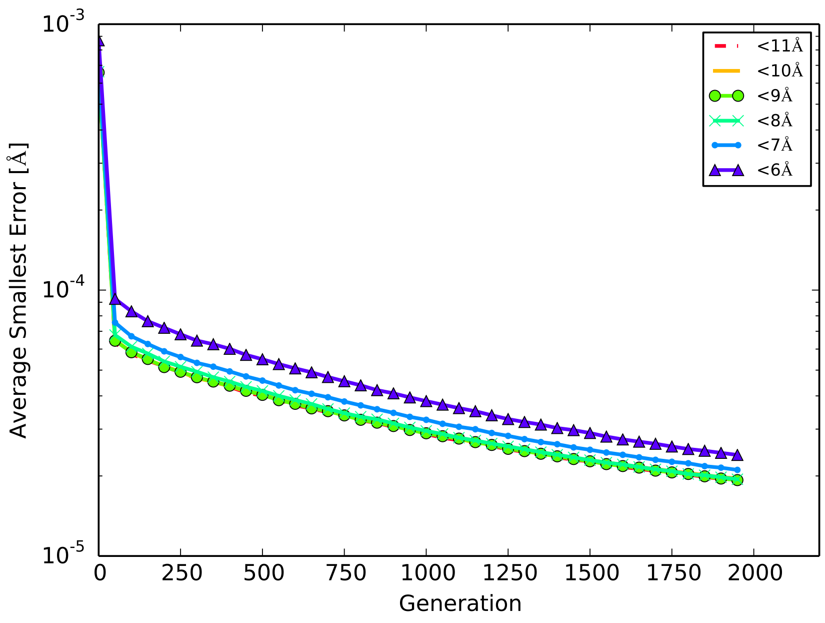


Figure 9.8: Average smallest error of solutions to the 3-peptide molecule obtained in different optimization processes with incomplete information of inter-atomic distances. Note that, by exploiting only distances $d_{ij} < 6$Å, MemHPG was able to reconstruct the 3D structure with an error lower than 10^{-4}Å with respect to the original structure.

Table 9.1: Results of the reconstruction of protein structures with MemHPG using only distances $d_{ij} < 6$Å or $d_{ij} < 7$Å.

		$d_{ij} < 6$Å		$d_{ij} < 7$Å	
PDB ID	$\mathfrak{N}$	ε [Å]	RMSD [Å]	ε [Å]	RMSD [Å]
1PTQ	402	0.152	1.23	0.019	0.08
1CTF	487	0.180	1.46	0.037	0.18
1RGD	548	0.149	1.24	0.014	0.04
1HOE	558	0.172	1.63	0.130	1.7
1LFB	641	0.206	2.21	0.254	2.08
1F39	767	0.278	3.25	0.090	0.93
1PHT	814	0.291	2.02	0.123	1.86
1POA	914	0.056	0.99	0.074	1.26
1AX8	1003	0.092	2.27	0.075	1.59

proteins 1AX8, 1HOE and 1CTF a perfect structural alignment was obtained; however, concerning protein 1F39, there is a slight discrepancy between the correct structure and the one obtained with MemHPG, probably due to an error in the reconstruction of a small portion of the protein (as better explained in the caption of Figure 9.9), while the overall structure is preserved also in the unaligned region.

These results are remarkable, since MemHPG only relies on (incomplete) distance matrices and general features of protein structures, while the other methods require additional *a priori* assumptions about proteins to achieve good results. Moreover, two additional advantages of this method reside in its intrinsic stochasticity and extensibility: on the one hand, the various reconstructed structures (with low error values) that can be obtained in each run of MemHPG are useful to represent the structural variability observed in biological molecules, which is a source of noise in NMR data; on the other hand, MemHPG can be easily improved by including a molecular force field in the scoring function during the final stages of the optimization process, in order to select structural models that are more realistic from a physical point of view.

Even though the parallel architecture accelerates the execution of MemHPG, an efficient non-sequential implementation of the crossover mechanism is far from trivial and currently under investigation. Since MemHPG relies on incomplete distance information, multiple runs of the method may yield a set of different optimal conformations: the GPU acceleration would allow to collect statistical information about the potential structures of the analyzed protein.

Further work is currently in progress to overcome the limitation of 1024 atoms per candidate protein, due to the mapping between protein atoms and CUDA blocks. This modification requires a deep restructuring of the code, since the indexing of atoms would be unaligned with respect to CUDA execution hierarchy. A secondary effect of this modification is that the efficient parallel reduction, used to assess the fitness value, could no longer be exploited, since threads belonging to different blocks cannot communicate and synchronize.

Finally, both the crossover operator and the chirality correction function are currently performed on the CPU. Distributed and more efficient algorithms are under investigation, in order to further reduce the computational complexity of MemHPG.

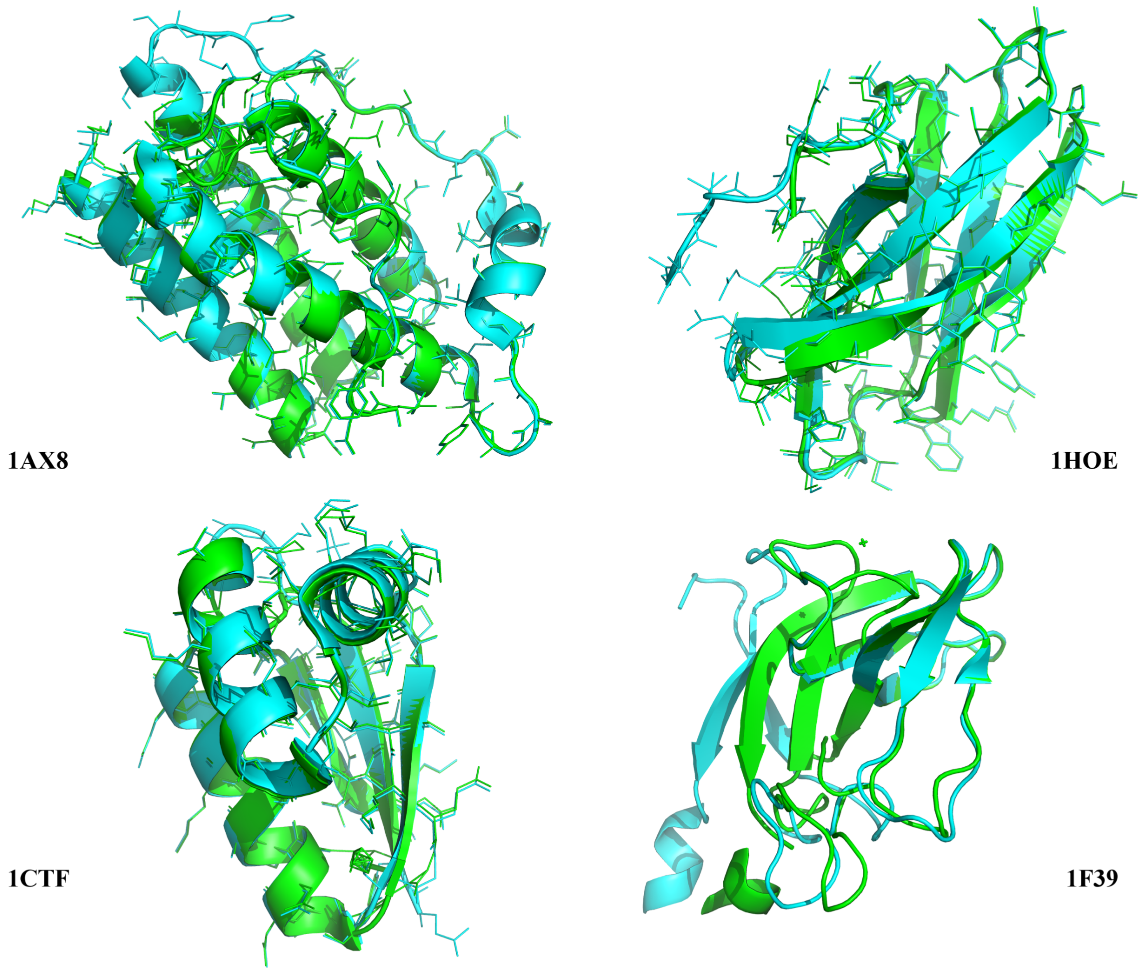


Figure 9.9: Examples of the structural alignment between the structures available in the PDB database (*cyan*) and protein structures reconstructed by MemHPG, using distance matrices **d** with $d_{ij} < 6\text{Å}$ (*green*). The alignments are correct, even though, in the case of protein 1F39, there is a slight discrepancy between the correct structure and the one obtained with MemHPG, probably due to an error in the reconstruction of a small portion of the protein connecting two major structural motifs, while the overall structure is preserved also in the unaligned region. This kind of errors can arise in portions of the proteins with extended structure, when a very low number of inter-atomic distances are available. Images and alignment obtained with PyMOL [221].

Chapter 10

Discussion

The goal of this thesis was to prove the feasibility of Evolutionary Computation and Swarm Intelligence to solve complex biological problems. To this aim, different methods were developed and successfully applied for parameter estimation of stochastic constants (cuPEPSO, Chapter 6), the reverse engineering of reaction-based models (cuRE, Chapter 7), the automatic design of gene regulation networks (cuGENED, Chapter 8) and the inference of protein structures (MemHPG, Chapter 9).

One of the main difficulties related to these problems and, in particular, to the corresponding computational methods, is the necessity of executing a large number of simulations or, in the case of protein inference, the simultaneous update of a large number of atoms positions. In order to reduce the relevant running time, all methods proposed in this thesis were accelerated using GPUs: the problems of PE, RE and ED rely on the parallel execution of the algorithm used to generate the systems dynamics (by means of cupSODA and cuTauLeaping, Chapter 5), while MemHPG was developed directly on the GPU (Chapter 9). Because of the GPU acceleration and the distributed approach of calculations, it is hard to compare the computational effort of the methodologies presented in this thesis with respect to the state-of-the-art methods.

10.1 A critical discussion of the proposed methods

In the case of PE, at the time of writing, cuPEPSO represents the only method which allows to consider target data collected in multiple laboratory experiments, and performed under different initial conditions. An advantage of this approach is that it yields a more robust estimation of the missing parameters, which avoids the over-fitting to a single experimental condition. cuPEPSO relies on a multi-swarm version of PSO (Section 3.3.2), in which each sub-population exploits the target data corresponding to a specific initial condition. The swarms periodically exchange the best particles and cooperate in identifying a common parameterization, able to fit the target data related to *all* initial conditions. The target data, used for the fitness evaluations, consists in a few samples of the chemical species, measured in multiple repetitions during each experiment, a scenario that represents the common experimental workflow used in biology laboratories for the analysis of cellular processes. The fitness evaluations are based on the comparison between the target data and the simulation outcome of the biochemical system, executed using the model parameterization encoded by each particle. Specifically, in cuPEPSO the simulation is performed by means of the tau-leaping stochastic simulation algorithm, in order to consider the intrinsic stochasticity of biochemical systems. Because of this peculiar structure, it is hard to compare the quality of the parameters optimization achieved by cuPEPSO with respect to other PE methods. Moreover, since no competing PE methodology exploits GPUs and almost no published works discuss the computational complexity of the proposed methods, it is also hard to carry out a direct comparison of the computational efficiency. Nevertheless, it can be claimed that cuPEPSO represents a more favorable methodology, at least with respect to alternative approaches for PE — especially those exploiting a probabilistic framework [205] — which require a huge computational effort.

Even in the case of efficient probabilistic non-parametric methodologies for PE like *Particle Filtering* (PF) [75], a large number of samples of the search space is required: Liu and Niranjan, for instance, showed that thousands particles combined with extremely dense time-series are necessary to converge to an optimal estimate [190], even in the case of a small set of unknown parameters. It is worth noting that both cupSODA and cuTauLeaping (Chapter 5) might be easily integrated into these alternative methodologies for the calculation of the likelihood of putative parameters. Besides the need for a large number of simulations, PF and similar techniques also have the drawback that a poor choice of the prior distribution prevents the algorithm from converging to an optimal estimation, an issue not affecting population-based algorithms.

The RE methodology proposed in this thesis, named cuRE, relies on a "two-level" approach, in which the network is inferred with CGP (Section 3.2.4) and kinetic parameters are estimated with PSO. Similarly to cuPEPSO, this RE methodology exploits sparsely sampled time-series, so that it can be used with common laboratory target data, where only a small number of time samples are usually measured. In Chapter 7, the effectiveness of cuRE was demonstrated, as it is able to infer a network of reactions that perfectly fits the target data. Thanks to the use of CGP, the output of cuRE is a RBM, based on mass-action kinetics, composed of a set of human-comprehensible parameterized chemical reactions. This is completely different from many alternative methods, in which the model is generally based on differential equations using arbitrary kinetics [326] or on approximate models like S-systems [59]. CGP also provides a means to control the bloating of the candidate solutions, without the need for parsimony terms in the fitness function (as instead used in, e.g., [241, 326]).

cuRE was tested on a model describing the synthesis and degradation of keton bodies, previously exploited to show the feasibility of an alternative methodology based on GP [166]. Although the RBM reverse engineered by cuRE was able to perfectly reproduce the target dynamics, it is structurally different from the target network: this issue is due to the *indistinguishability* of networks [327], an interesting topic that is further discussed in Section 10.3.

The computational effort required by alternative RE methods is usually not specified in literature, with the notable exception of the work presented in [167], where a cluster of 1000 machines was used to evolve a GP population of 100000 candidate solutions. cuRE, instead, exploits the GPU-powered simulator cupSODA [230], which is used to parallelize the fitness evaluations of the PE phase (Chapter 7). At the time of writing, cupSODA cannot handle the parallel simulations of more than one RBM at a time, because of the scarcity of high-performance memories (see Section 4.1), and thus it cannot currently simulate multiple CGP individuals. As a consequence, the computational complexity of cuRE scales linearly with the number of PSO particles that are used to determine the parameterization of the current CGP individual. To the aim of further reducing this complexity, a modified version of cupSODA, able to simulate multiple RBMs, is currently under development. In order to reduce the serialization due to conditional branches (see Section 4.1), caused by the different RBMs to be simulated, the multiple models will be organized in specific warps, that is, each warp will execute threads concerning a single model. Using this strategy, the complexity of the RE methodology could be reduced to $O(GEN_{\texttt{MAX}} \cdot IT_{\texttt{MAX}})$, achieving a result similar to the MIMD (Multiple Instruction Multiple Data) architecture that is usually adopted to distribute the calculations of single-population evolutionary algorithms [7]. It is worth noting that the complexity $O(GEN_{\texttt{MAX}} \cdot IT_{\texttt{MAX}})$ cannot be further reduced: each iteration of an EC or SI method, indeed, requires the results of the previous iteration.

Considering the ED problem, to the best of my knowledge no existing method for the automatic design of gene regulation networks defines the system at the level of detail of biochemical interactions, which is cuGENED's main novelty. Therefore, a direct comparison with other methods cannot be assessed; moreover, no competing ED method exploits any GPU acceleration. The outcome of cuGENED is a fully detailed mechanism of gene regulation, which cannot be clearly described by other formalisms as S-systems (Section 2.2.4), which work at a higher level of abstraction. Similarly to cuRE, cuGENED is able to infer gene regulatory networks fitting with the target data, even though the proposed models might then be hard to realize with the current laboratory techniques. Thus, a further process of refinement and simplification would be necessary, not to mention the issue of indistinguishability that affects also this "forward engineering" problem. Nevertheless, since cuGENED produces mechanistic descriptions of gene regulation, the role of genes and intermediate products is much more immediately understandable by synthetic biologists, than any mathematical model of gene regulatory networks inferred by other approaches.

Both cuRE and cuGENED rely on CGP, whereby biochemical networks are represented as fixed-length vectors of integer numbers. This representation is then mapped onto algebraic expressions which are ultimately interpreted as chemical reactions. This approach is robust and ensures the closure and sufficiency properties described in Section 3.2.3. Nevertheless, duplicate and not realistic chemical reactions could be inferred: although they can be consistent and formally correct, these reactions need to be identified and removed from the network. This process is computationally expensive, so that an alternative and more efficient approach should be employed. To this aim, in Section 7.2 a novel evolutionary methodology based on the Petri net formalism [53] was proposed: the Evolutionary Petri Net [229]. Although the EPNs framework is absolutely general and not tailored on the RE/ED problems, it allows to directly evolve a population of well-formed and realistic models of biochemical systems.

Concerning the MDGP problem for the inference of protein structures (Section 9.1), the proposed memetic/hybrid methodology (MemHPG) is absolutely competitive. MemHPG relies on partial inter-atomic distances data measured with NMR experiments, which are exploited by a hybrid memetic algorithm. The algorithm is hybrid in the sense that it combines PSO, a SI technique, with the crossover mechanism typical of GAs (see Section 3.2.1), so that the best protein substructure can be exchanged between two individuals. It is memetic (see Section 3.4) because, after each crossover, a gradient-based local search algorithm is exploited to identify the best roto-translation of the exchanged substructure, therefore minimizing the impact on the fitness function. After the memetic step, a chirality correction is finally applied to identify and fix the potential wrong symmetries of chiral atoms.

MemHPG is a heterogeneous algorithm, that is, it exploits more than one type of architecturally different computing units [34], in this case CPUs and GPUs. Specifically, the crossover mechanism, the chirality correction and the local search are performed on the CPU; the PSO-layer and the fitness evaluations are performed on the GPU. This heterogeneous implementation represents the current bottleneck of MemHPG and is responsible for the largest part of the overall running time: a new version, completely GPU-bound, is currently under development.

Differently from other methods for protein reconstruction, MemHPG does not require any *a priori* assumption about the molecule under analysis [129], it yields good results in the case of incomplete information [76] and it does not apply any simplification (e.g., discretization) to the problem [220]. In addition, differently from the large consumption of memory that is typical of branch and prune methods [173], MemHPG has a constant and extremely small memory footprint. MemHPG is a fast algorithm, reaching a high level of parallelism, since a thread is assigned to each atom of each candidate protein structure (see a schematization in Figure 9.5). In principle, MemHPG can be applied to any target protein; nevertheless, the current GPU implementation has a drawback with respect to other algorithms. As a matter of fact, CUDA limits the maximum size (in terms of atoms) of target proteins: since each candidate structure is assigned to a specific CUDA block, the maximum number of atoms is bounded to 1024 on the current CUDA architectures (see Table 4.1 for additional details). This limitation will be removed in the forthcoming implementation of MemHPG, which relies on a multi-block PSO-layer (i.e., more than 1024 threads can be assigned to the same candidate structure) and implicit synchronization mechanisms between blocks.

Fabry-Asztalos *et al.* proposed an alternative GPU-powered method for the MDGP with incomplete information [88], which is based on geometric build-up followed by a SA-based optimization. Even though the SA step is the only part that is accelerated on the GPU, the overall running time of the method is small. Unfortunately, because of MemHPG's limitation to 1024 atoms per candidate solution, a direct comparison of the two implementations is currently impossible. As a final remark, I underline here the importance of MemHPG in the whole workflow of *in silico* investigation of biochemical systems, since the inference of the structures performed can be a means to improve the PE performances. As a matter of fact, Molecular Dynamics methods are used to determine their affinity with ligands or enzymes. This information can be then considered either to directly estimate the kinetic parameters [70, 268, 319] or, at least, to restrict the search space of possible kinetic rates, providing an *a priori* distribution that can be exploited during the initialization of PSO particles.

Considering the GPU-powered simulation, cupSODA, coagSODA and cuTauLeaping outperform any existing sequential implementation of numerical integration methods and stochastic simulation algorithms when a large number of simulations needs to be executed. This is mainly due to the clock frequency of the SMXs, which is lower than the CPU. The slowness of the single core is counterbalanced by the large number of cores a SMX contains, and by the presence of multiple SMXs in a single GPU. Hence, a single algorithm run is faster on the CPU but, as the number of parallel threads increases, GPUs allow better performances.

The performances of cupSODA and cuTauLeaping depend on the availability of the high-performances memories (e.g., the shared memory and constant memory, see Section 4.1). Specifically, low-latency memory banks are used to store frequently accessed data (e.g., the states of the simulations), whereas cache-equipped memory banks are used to store the non-mutable data structures (e.g., the model). Unfortunately, the shared memory is a very limited resource: at the time of writing, up to 96 KB for each SMX can be used on the most advanced CUDA architectures (see Table 4.1). Therefore, the larger the system (in terms of bytes required to store the state of the simulation), the higher the shared memory footprint and the lower the number of simultaneous threads. By altering the memory indexing and pointers, the shared memory can be replaced by the global memory for the storage of all the states of the simulations, making registers the only limiting factor for GPU's usage. Although this strategy would significantly affect performances, it currently represents a feasible way for the analysis of large-scale systems, potentially avoiding the need of using alternative methods to reduce the number of simulations [311].

10.2 Towards a further automatization of inference and simulation methods

All the inference methodologies presented in this thesis share a common trait: they require the user to choose some functioning settings, which have a relevant impact to the performances. For instance, cuGENED (Section 8.2) requires the choice of the following values:

- population size I;
- number of particles n;
- mutation rate ρ;

- grid size (i.e., n_r and n_c);
- number of unknown intermediate complexes (i.e., $|\Sigma|$);
- number of output reactions n_o;
- maximum velocity of particles $v_{\mathtt{max}}$;
- social factor C_{soc};
- cognitive factor C_{cog};
- inertia weight w.

Since the methods proposed in this thesis are supposed to be used by modelers and computational biologists who might have poor or no knowledge about optimization algorithms, it is necessary to strongly reduce the number of free settings. Heuristics and default settings can be helpful to reduce this complexity, as discussed in Chapter 8, but they might not always represent the best choice leading to optimal performances. An alternative solution would be the use of self-adapting or settings-free algorithms, which completely remove the need for any functioning settings. All the methodologies described so far would benefit from the definition of a novel strategy for the automatic adaptation of the settings during the optimization phase. One example of such approach was proposed by Tomassini *et al.* in the context of GP [334]. This method dynamically changes the size of the population, according to the behavior of the population itself. Specifically, the population is decreased when the fitness of the best individual is improving, while new individuals are added when the GP enters a stagnation phase. It is worth noting that, in order to reduce bloating, the individuals with larger size are given a higher probability of being removed. This adaptive method — which could be used to automatically set the number of CPs in the CGP for both RE and ED — has two relevant advantages: first, it reduces the number of individuals and, consequently, the overall computational effort; second, it limits the size of the candidate solutions which, in the case of biochemical systems, implies a smaller set of reactions, which leads to faster simulations and a further reduced computational effort.

All the biochemical methodologies described in this thesis, including the simulation tools, are designed to exploit reaction-based modeling and MAK, which is the most general approach for the description of a chemical system. This design choice allows a great flexibility, thanks to the full decoupling between the simulation and the inference algorithms: any kind of simulator, indeed, can be used by the evolutionary algorithms and can be chosen according to the characteristics of the system under investigation (e.g., presence of biological noise). Again, to create completely automatic inference tools, it is necessary to avoid the need for the selection of a specific simulation algorithm. To this aim, a GPU-powered hybrid multi-scale simulator — similar to those described in Section 2.3.3 and based on a dynamic partitioning of reactions according to the molecular amounts and the propensity values — is currently under development [25].

MAK have a sound justification but, sometimes, alternative kinetics can be employed to describe a biochemical phenomenon. For instance, Hill kinetics is largely exploited to model the cooperative binding of ligands to macromolecules, which can be enhanced by the presence of other ligands on the molecule [226], as in the case of oxygen's binding to haemoglobin [132]. An extended version of cupSODA's parsing system (see Section 5.1), able to consider a general set of kinetics, is currently under development. In this context, for instance, the coagSODA simulator (described in Section 5.2) was developed for the specific purpose of investigating a model of the Blood Coagulation Cascade, defined in terms of both MAK and Hill functions.

Even though a general extension of cupSODA to integrate additional kinetics can be straightforwardly repeated for any kind of non-MAK models, the long-term goal is to create a "black-box" general-purpose GPU-powered biochemical simulator able to simulate any model specified by means of RBMs or ODEs.

10.3 The issue of indistinguishability

The *indistinguishability* of different networks (see, e.g., Sections 7.1.2 and 8.4) is an interesting topic in the context of the analysis of biochemical systems. Caused by an intrinsic problem affecting both RE and ED [63], it can be analyzed from multiple points of view. Conceptually, it means that a phenomenon can be explained by multiple alternative biochemical mechanisms. Thus, the RE problem cannot be solved without additional domain knowledge — collected either from existing databases or with novel experiments — or by relying on some mechanism for the automatic identification of the most likely reactions, possibly determined by means of logic inference.

Mathematically, indistinguishability can be explained by the simple example provided by Craciun and Pantea [63]. Consider the following two different sets of parameterized reactions:

$$\begin{aligned} R_1 &: S_0 \xrightarrow{2/9} S_1 + S_2 \qquad & R_1 &: S_0 \xrightarrow{5/9} S_1 + S_3 \\ R_2 &: S_0 \xrightarrow{1/6} 2S_1 & R_2 &: S_0 \xrightarrow{1/9} 2S_2 \\ R_3 &: S_0 \xrightarrow{11/18} 2S_3 & R_3 &: S_0 \xrightarrow{1/3} 2S_3 \end{aligned}$$

By assuming MAK, the biochemical networks are equivalent to the following two systems of coupled ODEs:

$$\begin{aligned} \frac{\mathrm{d}S_0}{\mathrm{d}t} &= -\frac{2}{9}S_0 - \frac{1}{6}S_0 - \frac{11}{18}S_0 = -\left(\frac{2}{9} + \frac{1}{6} + \frac{11}{18}\right) S_0 = -S_0 \\ \frac{\mathrm{d}S_1}{\mathrm{d}t} &= \frac{2}{9}S_0 + 2 \cdot \frac{1}{6}S_0 = \left(\frac{2}{9} + \frac{1}{3}\right) S_0 = \frac{5}{9}S_0 \\ \frac{\mathrm{d}S_2}{\mathrm{d}t} &= \frac{2}{9}S_0 \\ \frac{\mathrm{d}S_3}{\mathrm{d}t} &= 2 \cdot \frac{11}{8}S_0 = \frac{11}{9}S_0 \end{aligned}$$

for the first system and

$$\begin{aligned} \frac{\mathrm{d}S_0}{\mathrm{d}t} &= -\frac{5}{9}S_0 - \frac{1}{9}S_0 - \frac{1}{3}S_0 = -\left(\frac{5}{9} + \frac{1}{9} + \frac{1}{3}\right) S_0 = -S_0 \\ \frac{\mathrm{d}S_1}{\mathrm{d}t} &= \frac{5}{9}S_0 \\ \frac{\mathrm{d}S_2}{\mathrm{d}t} &= 2 \cdot \frac{1}{9}S_0 \\ \frac{\mathrm{d}S_3}{\mathrm{d}t} &= \frac{5}{9}S_0 + 2 \cdot \frac{1}{3} = \frac{11}{9}S_0 \end{aligned}$$

for the second system. It is clear that the two systems correspond to the same differential equations, so that the dynamics they generate are completely identical. Craciun and Pantea also proved that different systems characterized by the same dynamic behavior — named *confoundable* — are a consequence of the nonempty intersection of the convex cones generated by the state change vectors of the biochemical systems [63].

Epistemologically, the indistinguishability of networks has interesting implications. According to philosopher Popper, all equivalent theoretical models should be falsified in order to identify the correct explanation [266]. Falsification exploits the *modus tollens* of deductive logic so that, given a theory p and its consequence q, it follows:

$$\frac{p \rightarrow q \qquad \neg q}{\neg p}$$

This conceptual framework is more adequate than verification, since the inductive approach cannot help in discriminating between models having an identical behavior.

The Occam's razor — often used to discriminate between equivalent theories and implemented algorithmically by imposing a parsimony term on the fitness functions [241, 326] — is not adequate either, since nature does not necessarily evolve towards "simple" biochemical networks. As a matter of fact, it was shown that evolution generally produces complex systems characterized by increased robustness and error-tolerance [147].

The *fälschungsmöglichkeit* criterion [265] — that Popper proposed to distinguish between theories that "simply" confirm the observed phenomena from completely general explanations — is not always satisfiable by means of algorithmic approaches only. That is, the falsification of some proposition q can require the design of additional laboratory experiments. Unfortunately, in general, it is not trivial to design an *ad hoc* biochemical experiment to sustain the falsifiability of a hypothetic model.

For instance, the cuGENED algorithm (Chapter 8) not only derives a set of biochemical reactions able to produce a desired behavior, but also makes the inference of the number of unknown intermediate complexes produced by reactions. A possible way to falsify a solution proposed by cuGENED could be to investigate the exact number of different chemical species involved in the system, possibly analyzing the spectra coming from NMR data [113]. Vice versa, in the case of competing models characterized by the same number of chemical species, the falsification of the chemical reactions can be more tricky and potentially unfeasible because of experimental limitations.

Despite the plethora of problems caused to the RE methods by the possibility of reconstructing equivalent networks, indistinguishability has a philosophical relevance: which is the reason why biochemical systems evolved towards *one* specific network, among the multiple possible and equivalent possibilities? Arguably, they evolved from ancestor biochemical systems, with new reactions and functional modules gradually added up through the ages. These new features appeared because of random genetic mutations which introduced brand new or modified biomolecules. One example of such mechanisms is the *patchwork evolution* [175], which postulates that biomolecules (e.g., enzymes) initially having broad substrate specificity evolved a specialization through gene duplication, eventually producing two similar molecules that accept different substrates. This new genetic material managed to survive in the next generations because of some beneficial effect for the population, for instance the possibility to survive a depletion of one substrate. Thus, the evolutionary process which led to actual biochemical networks seems to be a sort of sub-optimal *greedy* process, constantly reusing vestigial biochemical material.

Chapter 11

Conclusions

This thesis presented four methods based on EC and SI to solve complex biological problems at different levels of granularity:

- the inference of protein structures, performed with a hybrid memetic algorithm named MemHPG, which combines GAs, PSO and a local search based on Gradient Descent;
- the PE of stochastic biochemical systems, performed with a multi-swarm PSO algorithm named cuPEPSO;
- the RE of biochemical systems, performed with a hybrid algorithm named cuRE, combining CGP and PSO;
- the ED of GRMs, performed with a hybrid algorithm named cuGENED, combining CGP and PSO.

The positive results shown in this thesis demonstrate the feasibility of EC and SI to solve of complex tasks in Systems Biology, Synthetic Biology and Computational Structural Biology.

Since all methods are population-based, they require a massive number of calculations for the fitness evaluations. In order to reduce their running times, all methodologies were implemented on GPUs, exploiting the CUDA architecture. Specifically, the PE, RE and ED methods embed two GPU-powered biochemical simulators developed for this purpose: cupSODA, a deterministic ODE solver based on the LSODA algorithm, and cuTauLeaping, a stochastic simulator based on the tau-leaping algorithm. Both simulators allow a strong acceleration with respect to the sequential counterpart and the performance gap between the CPUs and GPUs is expected to increase as more powerful GPUs are developed over the years.

These simulators are relevant not only to speed up the fitness evaluations, but they also represent valuable tools for the investigation of biochemical systems. Indeed, a strong reduction of the simulation time allows a deeper investigation of biological systems, for instance in the case of time consuming tasks relying on a massive number of simulations, like parameter sweep analysis [235] or sensitivity analysis [49]. A clear example of this advantage is the analysis of cAMP oscillations in the Ras/cAMP/PKA pathway: as it was shown in Figure 5.28, given a fixed amount of time, GPU parallelization allows to carry out a larger number of simulations with respect to the traditional CPU execution. Another example discussed in this thesis is represented by coagSODA (Section 5.2), the GPU-powered simulator specifically developed to investigate the blood coagulation cascade model, which allows a relevant 53× speedup with respect to the sequential execution of simulations, therefore allowing an in-depth analysis of perturbed conditions of this system [51].

Thanks to their reduced energy consumption, GPUs also allow to leverage the petascale performances of the most recent supercomputers: in October 2014, the second most powerful computer on the planet — Cray's Titan at the Oak Ridge National Laboratory [72] — is equipped with 18688 Nvidia Tesla K20X GPUs. Moreover, the first 15 entries of the Green500 list of the most power-efficient supercomputers[1] are accelerated by means of Nvidia Tesla K20 GPUs. The source code of cupSODA, coagSODA and cuTauLeaping is completely portable: it is ready to be compiled and scheduled for execution on these machines. As a matter of fact, at the time of writing, the coagSODA simulator is running on CINECA's EURORA GPU cluster[2], to carry out a thorough sensitivity analysis of the blood coagulation cascade model, requiring a total of 610560 simulations, which are distributed over 64 Tesla K20 GPUs.

As discussed in the previous chapters, all the simulation and inference methodologies will now undergo a further process of extension and optimization. Orthogonally to these improvements, there is a need for a unified Graphic User Interface (GUI). A future goal is to develop a comprehensive and user-friendly toolkit encompassing all the methods described in this thesis [28], which is supposed to provide computational biologists with the necessary tools for the definition of mathematical models, the inference of any missing information and the efficient execution of advanced analyses.

Besides these aspects, there is room for additional future developments, which are graphically schematized by the white hexagons in Figure 11.1 and described in what follows.

[1]Visit http://www.green500.org for further information.
[2]Visit http://www.cineca.it/en/content/eurora for further information.

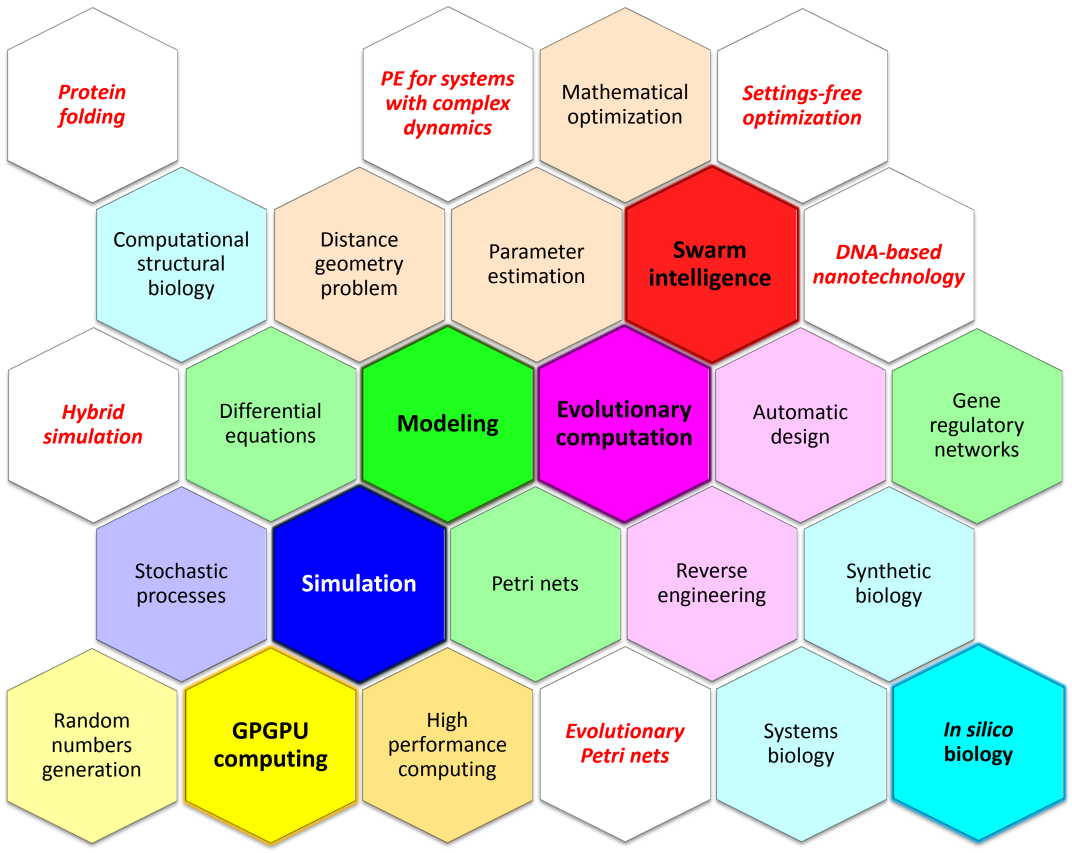


Figure 11.1: Schematization of some future developments of this thesis. The topics and achievements of the thesis are represented by the colored hexagons, while the white hexagons represent ongoing or future work. Hybrid approaches similar to MemHPG (Chapter 9) can be exploited for the problems of protein folding and structure prediction, according to a given amino-acids sequence. The PE method described in Chapter 6 — which represents the core of the RE and ED tools — can be extended to analyze systems characterized by complex dynamics. Settings-free versions of the EC and SI methodologies, which require no choices from the user, are under investigation. EC and SI can be exploited to create an automatic methodology for the development of DNA-based artifacts. A hybrid simulation algorithm is currently under development, to the aim of creating a multi-scale, “black-box” simulator accelerated with GPUs. Finally, EPNs (Chapter 7) will be applied to the RE and ED of biochemical systems. Eventually, all this methods will be integrated into a unified comprehensive toolkit.

Protein folding

As described in Chapter 10, the structure of macromolecules can be useful to estimate the kinetic parameters: this represents a bridge between Computational Structural Biology and Systems Biology, as the shapes of molecules become a means to determine the dynamic properties of the system they are involved in. A further bridge can be established between Bioinformatics and Structural Biology concerning the problem of *protein folding*, that is, the prediction of the protein's tertiary structure according to its primary structure. An extended version of MemHPG, which considers new constraints calculated according to amino-acids sequences, is under investigation.

PE for systems with complex dynamics

Biochemical systems can present complex dynamic behaviors (e.g., multi-stability [271], state-switching [2], oscillations [27]). For such systems, the "classic" fitness functions based on the comparison between the experimental time-series and the simulated dynamics cannot be employed in a straightforward way.

A different approach, able to consider most of these complex cases, could rely on the calculation of the fitness function defined as the distance between sampled and simulated distributions of the chemical species amounts. In the peculiar case of oscillatory systems, for instance, a fitness function based on the frequency-domain comparison could be conjectured.

Settings-free optimization

Some examples of settings-free optimization algorithms already exist, for instance TRIBES [61]. The strategies used in literature could represent a starting point for the conversion of the methodologies described in this thesis into completely automatic tools, so that the final user would not require any expertise nor the selection of any functioning settings. Thanks to this strategy, all methods would determine, or dynamically fine-tune, their best functioning settings, becoming totally automatic optimization tools able to perform the inference task by just entering the available target data.

DNA-based nanotechnology

DNA molecules have many interesting characteristics: the double-helical structure has a predictable geometry and the Watson-Crick pairing of bases allows a spontaneous self-assembly of complementary single-stranded sequences. Thus, by purposely designing the nucleotide sequences, DNA molecules have been exploited for the development of nano-scale artifacts with a variety of techniques [156, 171, 287]. In this context, EC can be applied to solve many complex problems, ranging from the optimal design of sequences [369], to the identification of the minimal set of "building blocks" [330] able to produce an arbitrary shape.

Hybrid simulation

The existing hybrid deterministic/stochastic multi-scale simulators, summarized in Section 2.3.3, could be improved to overcome some critical drawbacks [47]. For instance, the frequent re-partitioning into fast and slow reactions sets increases the quality of the simulation, but it is computationally expensive and usually causes a drop of performances. Moreover, to the best of my knowledge, there were no attempts to implement a GPU-powered version of any of the existing hybrid algorithms. This is probably due to the GPU's peculiar architecture: a challenging massive redesign of the algorithms would be necessary in order to fully leverage their computational power.

Evolutionary Petri nets

The EPNs described in Section 7.2 are supposed to provide both a better representation of putative biochemical interaction networks and more robust genetic operators for their evolution, with respect to traditional methods based on classic GP and CGP. This should improve the RE and ED methods from both the points of view of inference performances and computational efficiency, since EPNs allow the direct evolution of networks that are consistent with respect to the given assumptions. Moreover, EPNs allow to convey domain knowledge into the individuals and, by means of the set of hidden places, they allow to formulate new hypotheses about the complexes and the unknown species that are present in the system.

Even though EPNs were developed with RE and ED of biochemical systems in mind, they can be used to model, evolve and optimize any type of complex, distributed, asynchronous and concurrent system. As a matter of fact, Petri nets have been applied in a plethora of applications and disciplines [223]. In all these contexts — provided the definition of suitable fitness functions — EPNs can assist or completely replace the human intervention during the design and optimization of the systems.

Of course, the unified toolkit under development will embed all these methodologies.

*

* *

The conjunction of high-throughput methodologies, providing a huge amount of quantitative data, and the availability of an unprecedented computational power is creating the context for the development and the analysis of detailed mechanisms-based models of biological systems, allowing novel and exciting *in silico* investigations. The predictive power of this approach will permit new breakthroughs and paradigm shifts in Systems Biology, Synthetic Biology and Computational Structural Biology. This Ph.D. thesis represented an effort in providing scientists with novel methodologies in the aforementioned fields — able to fully leverage the cutting-edge technology — allowing the efficient execution of simulations and the possibility of automatically inferring missing information in the domain knowledge, in order to create increasingly complex models, towards the final goal of comprehending cellular complexity through whole cell simulation.

Bibliography

[1] A. M. Abdelbar, S. Abdelshahid, and D. C. Wunsch. Fuzzy PSO: a generalization of particle swarm optimization. In *Proceedings of the 2005 IEEE International Joint Conference on Neural Networks*, volume 2, pages 1086–1091. IEEE, 2005.

[2] M. Acar, J. T. Mettetal, and A. van Oudenaarden. Stochastic switching as a survival strategy in fluctuating environments. *Nature Genetics*, 40(4):471–475, 2008.

[3] J. Ackermann, P. Baecher, T. Franzel, M. Goesele, and K. Hamacher. Massively-parallel simulation of biochemical systems. In *Proceedings of Massively Parallel Computational Biology on GPUs, Jahrestagung der Gesellschaft für Informatik e.V*, pages 739–750, 2009.

[4] T. Aho, H. Almusa, J. Matilainen, A. Larjo, P. Ruusuvuori, K.-L. Aho, T. Wilhelm, H. Lähdesmäki, A. Beyer, M. Harju, S. Chowdhury, K. Leinonen, C. Roos, and O. Yli-Harja. Reconstruction and validation of RefRec: a global model for the yeast molecular interaction network. *PLoS ONE*, 5(5):e10662, 2010.

[5] H. Akaike. Likelihood of a model and information criteria. *Journal of Econometrics*, 16(1):3–14, 1981.

[6] T. Akutsu, S. Miyano, and S. Kuhara. Identification of genetic networks from a small number of gene expression patterns under the Boolean network model. In *Pacific Symposium on Biocomputing*, volume 4, pages 17–28, 1999.

[7] E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, 2002.

[8] L. Alberghina and H. V. Westerhoff. *Systems biology: Definitions and perspectives*, volume 13 of *Topics in Current Genetics*. Springer-Verlag, 2005.

[9] B. B. Aldridge, J. M. Burke, D. A. Lauffenburger, and P. K. Sorger. Physicochemical modelling of cell signalling pathways. *Nature Cell Biology*, 8:1195–1203, 2006.

[10] H. Alla and R. David. Continuous and hybrid Petri nets. *Journal of Circuits, Systems, and Computers*, 8(1):159–188, 1998.

[11] F. Amara, R. Colombo, P. Cazzaniga, D. Pescini, A. Csikász-Nagy, M. Muzi Falconi, D. Besozzi, and P. Plevani. *In vivo* and *in silico* analysis of PCNA ubiquitylation in the activation of the Post Replication Repair pathway in *S. cerevisiae*. *BMC Systems Biology*, 7(1):24, 2013.

[12] G. An, Q. Mi, J. Dutta-Moscato, and Y. Vodovotz. Agent-based models in translational systems biology. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, 1(2):159–171, 2009.

[13] S. Ando, E. Sakamoto, and H. Iba. Evolutionary modeling and inference of gene network. *Information Sciences*, 145(3):237–259, 2002.

[14] S. S. Andrews, T. Dinh, and A. P. Arkin. Stochastic models of biological processes. In *Encyclopedia of Complexity and Systems Science*, pages 8730–8749. Springer-Verlag, 2009.

[15] M. A. Arbab, G. M. Khan, and A. M. Sahibzada. Cardiac arrhythmia classification using cartesian genetic programming evolved artificial neural network. *Experimental & Clinical Cardiology*, 20(9):5334–5348, 2014.

[16] A. Arkin and J. Ross. Statistical construction of chemical reaction mechanisms from measured time-series. *Journal of Physical Chemistry*, 99(3):970–979, 1995.

[17] M. S. Arumugam and M. V. C. Rao. On the improved performances of the particle swarm optimization algorithms with adaptive parameters, cross-over operators and root mean square (RMS) variants for computing optimal control of a class of hybrid systems. *Journal of Applied Soft Computing*, 8(1):324–336, 2008.

[18] J. Ashworth, E. J. Wurtmann, and N. S. Baliga. Reverse engineering systems models of regulation: discovery, prediction and mechanisms. *Current Opinion in Biotechnology*, 23(4):598–603, 2012.

[19] T. Bäck. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, volume 1, pages 57–62. IEEE, 1994.

[20] P. Ballarini, R. Guido, T. Mazza, and D. Prandi. Taming the complexity of biological pathways through parallel computing. *Briefings in Bioinformatics*, 10(3):278–288, 2009.

[21] A.-L. Barabási and Z. N. Oltvai. Network biology: understanding the cell's functional organization. *Nature Reviews Genetics*, 5(2):101–113, 2004.

[22] M. Bellini, D. Besozzi, P. Cazzaniga, G. Mauri, and Nobile M. S. Simulation and analysis of the blood coagulation cascade accelerated on GPU. In *Proceedings of 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2014), Turin, Italy*, pages 590–593. IEEE Computer Society Conference Publishing Services (CPS), 2014.

[23] S. A. Benner and A. M. Sismour. Synthetic biology. *Nature Reviews Genetics*, 6(7):533–543, 2005.

[24] D. Besozzi. Computational methods in systems biology: Case studies and biological insights. In I. Petre, editor, *Proceedings of the 4th International Workshop on Computational Models for Cell Processes*, volume 116 of *Electronic Proceedings in Theoretical Computer Science*, pages 3–10, 2013.

[25] D. Besozzi, G. Caravagna, P. Cazzaniga, M. S. Nobile, D. Pescini, and A. Re. GPU-powered simulation methodologies for biological systems. In A. Graudenzi, G. Caravagna, G. Mauri, and M. Antoniotti, editors, Proceedings of *Wivace 2013 - Italian Workshop on Artificial Life and Evolutionary Computation*, volume 130 of *Electronic Proceedings in Theoretical Computer Science*, pages 87–91, 2013.

[26] 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 C. Pizzuti, M. D. Ritchie, and M. Giacobini, editors, *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics. 7th European Conference, EvoBIO 2009 Tübingen, Germany, April 15-17, 2009 Proceedings*, Lecture Notes in Computer Science, pages 116–127. Springer-Verlag, 2009.

[27] D. Besozzi, P. Cazzaniga, D. Pescini, G. Mauri, S. Colombo, and E. Martegani. The role of feedback control mechanisms on the establishment of oscillatory regimes in the Ras/cAMP/PKA pathway in *S. cerevisiae*. *EURASIP Journal on Bioinformatics and Systems Biology*, 1(10), 2012.

[28] D. Besozzi, M. S. Nobile, P. Cazzaniga, D. Cipolla, and G. Mauri. From the inference of molecular structures to the analysis of emergent cellular dynamics: accelerating the computational study of biological systems with GPUs. In *Proceedings of the NETTAB 2014 Workshop: from Structural Bioinformatics to Integrative Systems Biology*, pages 88–90, 2014.

[29] H. Beyer and H. Schwefel. Evolution strategies - a comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

[30] H.-G. Beyer. *The Theory of Evolution Strategies*. Springer-Verlag, 2001.

[31] W. J. Blake, M. Kærn, C. R. Cantor, and J. J. Collins. Noise in eukaryotic gene expression. *Nature*, 422(6932):633–637, April 2003.

[32] J. Born. *Evolutionsstrategien zur numerischen Lösung von Adaptationsaufgaben*. PhD thesis, Humboldt–Universität, Berlin, 1978.

[33] J. M. Bower and H. Bolouri. *Computational Modeling of Genetic and Biochemical Networks*. MIT Press, 2001.

[34] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik, and O. O. Storaasli. State-of-the-art in heterogeneous computing. *Scientific Programming*, 18(1):1–33, 2010.

[35] F. J. Bruggeman and H. V. Westerhoff. The nature of systems biology. *Trends in Microbiology*, 15(1):45–50, 2007.

[36] K. Brummel-Ziedins. Models for thrombin generation and risk of disease. *Journal of Thrombosis and Haemostasis*, 11(s1):212–223, 2013.

[37] K. E. Brummel-Ziedins, S. J. Everse, K. G. Mann, and T. Orfeo. Modeling thrombin generation: plasma composition based approach. *Journal of Thrombosis and Thrombolysis*, 37(1):32–44, 2014.

[38] A. P. Burgard and C. D. Maranas. Probing the performance limits of the *Escherichia coli* metabolic network subject to gene additions or deletions. *Biotechnology and Bioengineering*, 74(5):364–375, 2001.

[39] F. J. Burkowski. *Structural Bioinformatics: An Algorithmic Approach.* Chapman & Hall/CRC, 2008.

[40] K. Burrage, M. Hegland, F. MacNamara, and B. Sidje. A Krylov-based finite state projection algorithm for solving the Chemical Master Equation arising in the discrete modelling of biological systems. In A.N. Langville and W.J. Stewart, editors, *Proceedings of the Markov 150th Anniversary Conference*, pages 21–38, 2006.

[41] J. C. Butcher. *Numerical Methods for Ordinary Differential Equations.* John Wiley & Sons, 2003.

[42] S. Cagnoni, L. Vanneschi, A. Azzini, and A. G. B. Tettamanzi. A critical assessment of some variants of particle swarm optimization. In M. Giacobini et al., editor, *Applications of Evolutionary Computing. EvoWorkshops 2008: EvoCOMNET, EvoFIN, EvoHOT, EvoIASP, EvoMUSART, EvoNUM, EvoSTOC, and EvoTransLog, Naples, Italy, March 26-28, 2008. Proceedings*, Lecture Notes in Computer Science, pages 565–574. Springer-Verlag, 2008.

[43] T. Çakır, M. M. Hendriks, J. A. Westerhuis, and A. K. Smilde. Metabolic network discovery through reverse engineering of metabolome data. *Metabolomics*, 5(3):318–329, 2009.

[44] H. Cao, L. Kang, Y. Chen, and J. Yu. Evolutionary modeling of systems of ordinary differential equations with genetic programming. *Genetic Programming and Evolvable Machines*, 1:309–337, 2000.

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

[46] Y. Cao, D. T. Gillespie, and L. R. Petzold. Efficient step size selection for the tau-leaping simulation method. *The Journal of Chemical Physics*, 124(4):044109, 2006.

[47] G. Caravagna, P. Cazzaniga, M. S. Nobile, D. Pescini, and A. Re. Enhancing simulations of chemical reactions at mesoscales. In *BITS 2014 - 11th Annual Meeting of the Bioinformatics Italian Society*, 2014.

[48] S. Carrillo, J. Siegel, and X. Li. A control-structure splitting optimization for GPGPU. In *Proceedings of the 6th ACM conference on Computing frontiers*, CF '09, pages 147–150. ACM, 2009.

[49] P. Cazzaniga, R. Colombo, M. S. Nobile, D. Pescini, G. Mauri, and D. Besozzi. GPU-powered sensitivity analysis and parameter estimation of a reaction-based model of the Post Replication Repair pathway in yeast. In R. Autio, I. Shmulevich, K. Strimmer, C. Wiuf, S. Sarbu, and O. Yli-Harja, editors, *Proceedings of the 10th International Workshop on Computational Systems Biology (WCSB)*, volume 63, pages 109–110, 2013.

[50] P. Cazzaniga, C. Damiani, D. Besozzi, R. Colombo, M. S. Nobile, D. Gaglio, D. Pescini, S. Molinari, G. Mauri, L. Alberghina, and M. Vanoni. Computational strategies for a system-level understanding of metabolism. *Metabolites*, 2014, under revision.

[51] P. Cazzaniga, M. S. Nobile, D. Besozzi, M. Bellini, and G. Mauri. Massive exploration of perturbed conditions of the blood coagulation cascade through GPU parallelization. *BioMed Research International*, 2014, 2014. Article ID 863298.

[52] 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. *Journal of Biotechnology*, 133(3):377–385, 2008.

[53] C. Chaouiya. Petri net modelling of biological networks. *Briefings in Bioinformatics*, 8(4):210–219, 2007.

[54] A. Chatterjee and P. Siarry. Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization. *Computers & Operations Research*, 33(3):859–871, 2006.

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

[56] M. S. Chatterjee, W. S. Denney, H. Jing, and S. L. Diamond. Systems biology of coagulation initiation: Kinetics of thrombin generation in resting and activated human blood. *PLoS Computational Biology*, 6(9):e1000950, 2010.

[57] D. Cho, K. Cho, and B. Zhang. Identification of biochemical networks by S-tree based genetic programming. *Bioinformatics*, 22(13):1631–1640, 2006.

[58] Y. J. Cho, N. Ramakrishnan, and Y. Cao. Reconstructing chemical reaction networks: data mining meets system identification. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 142–150, 2008.

[59] I. C. Chou and E. O. Voit. Recent developments in parameter estimation and structure identification of biochemical and genomic systems. *Mathematical Biosciences*, 219(2):57–83, 2009.

[60] L. A. Chylek, L. A. Harris, C.-S. Tung, J. R. Faeder, C. F. Lopez, and W. S. Hlavacek. Rule-based modeling: a computational approach for studying biomolecular site dynamics in cell signaling systems. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, 6(1):13–36, 2014.

[61] M. Clerc. *Particle Swarm Optimization*. ISTE. Wiley, 2010.

[62] R. W. Colman, J. Hirsh, V. J. Marder, A. W. Clowes, and J. N. George, editors. *Hemostasis and thrombosis: basic principles and clinical practice*. Lippincott Williams & Wilkins, Philadelphia, USA, 5th edition, 2006.

[63] G. Craciun and C. Pantea. Identifiability of chemical reaction networks. *Journal of Mathematical Chemistry*, 44(1):244–259, 2008.

[64] G. Craciun, Y. Tang, and M. Feinberg. Understanding bistability in complex enzyme-driven reaction networks. *Proceedings of the National Academy of Sciences*, 103(23):8697–8702, 2006.

[65] G. M. Crippen and T. F. Havel. Distance geometry and molecular conformation. *Journal of Computational Chemistry*, 11(2):265–266, 1990.

[66] C. Damiani and P. Lecca. Model identification using correlation-based inference and transfer entropy estimation. In *UKSim Fifth European Symposium on Computer Modeling and Simulation*, pages 129–134. IEEE, 2011.

[67] G. B. Dantzig and M. N. Thapa. *Linear Programming 1: Introduction*, volume 1. Springer-Verlag, 1997.

[68] A. C. Deckard, F. T. Bergmann, and H. M. Sauro. Enumeration and online library of mass-action reaction networks. *ArXiv e-prints*, 2009.

[69] K. A. DeJong. *Evolutionary Computation: A Unified Approach.* The MIT Press, 2006.

[70] D. Dell'Orco. Fast predictions of thermodynamics and kinetics of protein–protein recognition from structures: from molecular design to systems biology. *Molecular BioSystems*, 5(4):323–334, 2009.

[71] L. Demattè and D. Prandi. GPU computing for systems biology. *Briefings in Bioinformatics*, 11(3):323–33, 2010.

[72] T. Diede, C. F. Hagenmaier, G. S. Miranker, J. J. Rubinstein, and W. S. Worley Jr. The Titan graphics supercomputer architecture. *Computer*, 21(9):13–30, 1988.

[73] K. A Dill, S. B. Ozkan, M. S. Shell, and T. R. Weikl. The protein folding problem. *Annual Review of Biophysics*, 37:289–316, 2008.

[74] L. Dipasquale, G. d'Ippolito, and A. Fontana. Capnophilic lactic fermentation and hydrogen synthesis by *Thermotoga neapolitana*: An unexpected deviation from the dark fermentation model. *International Journal of Hydrogen Energy*, 39(10):4857–4862, 2014.

[75] P. M. Djuric, J. H. Kotecha, J. Zhang, Y. Huang, T. Ghirmai, M. F. Bugallo, and J. Miguez. Particle filtering. *Signal Processing Magazine, IEEE*, 20(5):19–38, 2003.

[76] Q. Dong and Z. Wu. A linear-time algorithm for solving the molecular distance geometry problem with exact inter-atomic distances. *Journal of Global Optimization*, 22(1-4):365–375, 2002.

[77] M. Dorigo, E. Bonabeau, and G. Theraulaz. Ant algorithms and stigmergy. *Future Generation Computer Systems*, 16(8):851–871, 2000.

[78] A. Dräger, M. Kronfeld, M. J. Ziller, J. Supper, H. Planatscher, and J. B. Magnus. Modeling metabolic networks in *C. glutamicum*: a comparison of rate laws in combination with various parameter optimization strategies. *BMC Systems Biology*, 3(5), 2009.

[79] R. M. D'Souza, M. Lysenko, and K. Rahmani. Sugarscape on steroids: simulating over a million agents at interactive rates. In *Proceedings of the Agent 2007 Conference on Complex Interaction and Social Emergence*, page 7, 2007.

[80] X. Du, J. Wang, H. Zhu, L. Rinaldo, K. Lamar, A. C. Palmenberg, C. Hansel, and C. M. Gomez. Second cistron in *CACNA1A* gene encodes a transcription factor mediating cerebellar development and SCA6. *Cell*, 154(1):118–133, 2013.

[81] A. Eiben, C. van Kemenade, and J. Kok. Orgy in the computer: Multi-parent reproduction in genetic algorithms. In F. Moran, A. Moreno, J. J. Merelo, and P. Chacon, editors, *Advances in Artificial Life, Third European Conference on Artificial Life, Granada, Spain, June 4-6, 1995, Proceedings*, volume 929 of *Lecture Notes in Artificial Intelligence*, pages 934–945. Springer-Verlag, 1995.

[82] A. Eldar and M. B. Elowitz. Functional roles for noise in genetic circuits. *Nature*, 467(7312):167–173, 2010.

[83] J. Elf and M. Ehrenberg. Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases. *Systems Biology*, 1(2):230–236, 2004.

[84] M. B. Elowitz and S. Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, 2000.

[85] M. B. Elowitz, A. J. Levine, E. D. Siggia, and P. S. Swain. Stochastic gene expression in a single cell. *Science*, 297(5584):1183–1186, 2002.

[86] F. Emmert-Streib, G. V. Glazko, G. Altay, and R. de Matos Simoes. Statistical inference and reverse engineering of gene regulatory networks from observational expression data. *Frontiers in Genetics*, 3(8):1–15, 2012.

[87] B. H. Esteridge, A. P. Reynolds, and N. J. Walters. *Basic Medical Laboratory Techniques.* Cengage Learning, 4th edition, 2000.

[88] L. Fabry-Asztalos, I. Lorentz, and R. Andonie. Molecular distance geometry optimization using geometric build-up and evolutionary techniques on GPU. In *2012 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, pages 321–328, 2012.

[89] R. M. Farber. Topical perspective on massive threading and parallelism. *Journal of Molecular Graphics and Modelling*, 30:82–89, 2011.

[90] N. Fedoroff and W. Fontana. Small numbers of big molecules. *Science*, 297(5584):1129–1131, 2002.

[91] F. Fernández, M. Tomassini, and L. Vanneschi. Studying the influence of communication topology and migration on distributed genetic programming. In J. Miller et al., editor, *Genetic Programming. 4th European Conference, EuroGP 2001 Lake Como, Italy, April 18–20, 2001 Proceedings*, Lecture Notes in Computer Science, pages 51–63. Springer-Verlag, 2001.

[92] R. Fletcher. *Practical Methods of Optimization.* John Wiley & Sons, 2013.

[93] N. C. Garbett and J. B. Chaires. Binding: a polemic and rough guide. *Methods in Cell Biology*, 84:1–23, 2008.

[94] T. S. Gardner, C. R. Cantor, and J. J. Collins. Construction of a genetic toggle switch in *Escherichia coli. Nature*, 403(6767):339–342, 2000.

[95] C. Garmendia-Torres, A. Goldbeter, and M. Jacquet. Nucleocytoplasmic oscillations of the yeast transcription factor Msn2: Evidence for periodic PKA activation. *Current Biology*, 17(12):1044–9, 2007.

[96] J. E. Gentle. *Random Number Generation and Monte Carlo Methods.* Statistics and Computing. Springer-Verlag, 2nd edition, 2003.

[97] S. Ghaemmaghami, W. K. 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(6959):671–672, 2003.

[98] G. Gibson. Microarray analysis. *PLoS Biology*, 1(1):e15, 2003.

[99] M. A. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *The Journal of Physical Chemistry A*, 104(9):1876–1889, 2000.

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

[101] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Computational Physics*, 81:2340–2361, 1977.

[102] D. T. Gillespie. A rigorous derivation of the chemical master equation. *Physica A*, 188(1):404–425, 1992.

[103] D. T. Gillespie. The chemical Langevin equation. *The Journal of Chemical Physics*, 113(1):297–306, 2000.

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

[105] D. T. Gillespie. Stochastic simulation of chemical kinetics. *Annual Review of Physical Chemistry*, 58:35–55, 2007.

[106] L. Glass. Classification of biological networks by their qualitative dynamics. *Journal of Theoretical Biology*, 54(1):85–107, 1975.

[107] L. Glass and S. A. Kauffman. The logical analysis of continuous, non-linear biochemical control networks. *Journal of Theoretical Biology*, 39(1):103–129, 1973.

[108] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[109] D. E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3(5):493–530, 1989.

[110] P. J. E. Goss and J. Peccoud. Quantitative modeling of stochastic systems in molecular biology by using stochastic Petri nets. *Proceedings of the National Academy of Sciences*, 95(12):6750–6755, 1998.

[111] A. W. Götz, T. Wölfle, and R. C. Walker. Quantum chemistry on graphics processing units. *Annual Reports in Computational Chemistry*, 6:21–35, 2010.

[112] K. Gray. *Microsoft DirectX 9 Programmable Graphics Pipeline*. Microsoft Press, 2003.

[113] J. L. Griffin. Metabonomics: NMR spectroscopy and pattern recognition analysis of body fluids and tissues for characterisation of xenobiotic toxicity and disease diagnosis. *Current Opinion in Chemical Biology*, 7(5):648–654, 2003.

[114] A. Grosso, M. Locatelli, and F. Schoen. Solving molecular distance geometry problems by global optimization algorithms. *Computational Optimization and Applications*, 43(1):23–37, 2009.

[115] R. Gunawan, Y. Cao, L. Petzold, and F. J. Doyle. Sensitivity analysis of discrete stochastic systems. *Biophysical Journal*, 88:2530–2540, 2005.

[116] W. J. Gutjahr. ACO algorithms with guaranteed convergence to the optimal solution. *Information Processing Letters*, 82(3):145–153, 2002.

[117] A. C. Guyton and J. E. Hall. *Textbook of Medical Physiology.* Saunders, 2010.

[118] N. Hansen, D. V. Arnold, and A. Auger. Evolution strategies. In J. Kacprzyk and W. Pedrycz, editors, *Handbook of Computational Intelligence.* Springer-Verlag, in press.

[119] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of 1996 IEEE International Conference on Evolutionary Computation*, pages 312–317. IEEE, 1996.

[120] S. Harding, J. Leitner, and J. Schmidhuber. Cartesian genetic programming for image processing. In R. Riolo, E. Vladislavleva, M. D. Ritchie, and J. H. Moore, editors, *Genetic Programming Theory and Practice X*, pages 31–44. Springer-Verlag, 2013.

[121] A. B. Harris, R. D. Kamien, and T. C. Lubensky. Molecular chirality and chiral parameters. *Reviews of Modern Physics*, 71(5):1745–1757, 1999.

[122] L. A. Harris and P. Clancy. A "partitioned leaping" approach for multiscale modeling of chemical reaction dynamics. *The Journal of Chemical Physics*, 125(14):144107, 2006.

[123] M. J. Harvey and G. De Fabritiis. A survey of computational molecular science using graphics processing units. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 2(5):734–742, 2012.

[124] M. J. Harvey, G. Giupponi, and G. De Fabritiis. ACEMD: accelerating biomolecular dynamics in the microsecond time scale. *Journal of Chemical Theory and Computation*, 5(6):1632–1639, 2009.

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

[126] T. F. Havel. Distance geometry: Theory, algorithms and chemical applications. In *Encyclopedia of Computational Chemistry*, pages 723–742. John Wiley & Sons, 1998.

[127] M. Heinemann and S. Panke. Synthetic biology—putting engineering into biology. *Bioinformatics*, 22(22):2790–2799, 2006.

[128] A. Hellander. Efficient computation of transient solutions of the Chemical Master Equation based on uniformization and quasi-Monte Carlo. *The Journal of Chemical Physics*, 128(15):154109, 2008.

[129] B. Hendrickson. The molecule problem exploiting structure in global optimization. *SIAM Journal on Optimization*, 5(4):835–857, 1995.

[130] F. Herrera, M. Lozano, and A. M. Sánchez. A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study. *International Journal of Intelligent Systems*, 18(3):309–338, 2003.

[131] F. Herrera, M. Lozano, and J. L. Verdegay. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review*, 12(4):265–319, 1998.

[132] A. V. Hill. The combinations of haemoglobin with oxygen and with carbon monoxide. I. *Biochemical Journal*, 7(5):471, 1913.

[133] D. R. C. Hill, C. Mazel, J. Passerat-Palmbach, and M. K. Traore. Distribution of random streams for simulation practitioners. *Concurrency and Computation: Practice and Experience*, 25(10):1427–1442, 2013.

[134] W. S. Hlavacek, J. R. Faeder, M. L. Blinov, A. S. Perelson, and B. Goldstein. The complexity of complexes in signal transduction. *Biotechnology and Bioengineering*, 84(7):783–794, 2003.

[135] W. S. Hlavacek, J. R. Faeder, M. L. Blinov, R. G. Posner, M. Hucka, and W. Fontana. Rules for modeling signal-transduction systems. *Science Signaling*, 2006(344):re6, 2006.

[136] M. F. Hockin, K. C. Jones, S. J. Everse, and K. G. Mann. A model for the stoichiometric regulation of blood coagulation. *Journal of Biological Chemistry*, 277(21):18322–18333, 2002.

[137] J. H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, Michigan, USA, 1975.

[138] B. Hölldobler and E. O. Wilson. *The Superorganism: The Beauty, Elegance, and Strangeness of Insect Societies.* WW Norton & Company, 2008.

[139] L. Hong and J. Lei. Scaling law for the radius of gyration of proteins and its dependence on hydrophobicity. *Journal of Polymer Science Part B: Polymer Physics*, 47(2):207–214, 2009.

[140] S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer. COPASI - a COmplex PAthway SImulator. *Bioinformatics*, 22:3067–3074, 2006.

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

[142] H. Iba. Bagging, boosting, and bloating in genetic programming. In W. Banzhaf et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1053–1060. Morgan Kaufmann, 1999.

[143] H. Iba. Inference of differential equation models by genetic programming. *Information Sciences*, 178(23):4453–4468, 2008.

[144] T. Ideker, T. Galitski, and L. Hood. A new approach to decoding life: systems biology. *Annual Review of Genomics and Human Genetics*, 2:343–372, 2001.

[145] T. Jahnke and W. Huisinga. Solving the Chemical Master Equation for monomolecular reaction systems analytically. *Journal of Mathematical Biologyl*, 54(1):1–26, 2007.

[146] P. V. Jenkins, O. Rawley, O. P. Smith, and J. S. O'Donnell. Elevated factor VIII levels and risk of venous thrombosis. *British Journal of Hematology*, 157(6):653–663, 2012.

[147] H. Jeong, B. Tombor, R. Albert, Z. N. Oltvai, and A.-L. Barabási. The large-scale organization of metabolic networks. *Nature*, 407(6804):651–654, 2000.

[148] E. Jones, T. Oliphant, and P. Peterson. SciPy: Open source scientific tools for Python, 2001–.

[149] D. Karaboga and B. Akay. A comparative study of artificial bee colony algorithm. *Applied Mathematics and Computation*, 214(1):108–132, 2009.

[150] D. Karaboga and B. Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global Optimization*, 39(3):459–471, 2007.

[151] D. Karaboga and B. Gorkemli. A combinatorial artificial bee colony algorithm for traveling salesman problem. In *Innovations in Intelligent Systems and Applications (INISTA), 2011 International Symposium on*, pages 50–53. IEEE, 2011.

[152] D. Kartson, G. Balbo, S. Donatelli, G. Franceschinis, and G. Conte. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, Inc., 1994.

[153] H. Katagiri, K. Hirasawa, J. Hu, and J. Murata. Comparing some graph crossover in genetic network programming. In *Proceedings of the 41st SICE Annual Conference*, volume 2, pages 1263–1268. IEEE, 2002.

[154] S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3):437–467, 1969.

[155] S. A. Kauffman. The large scale structure and dynamics of gene control circuits: An ensemble approach. *Journal of Theoretical Biology*, 44(1):167–190, 1974.

[156] Y. Ke, L. L. Ong, W. M. Shih, and P. Yin. Three-dimensional structures self-assembled from DNA bricks. *Science*, 338(6111):1177–1183, 2012.

[157] J. Kennedy and R.C. Eberhart. Particle Swarm Optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[158] J. Kennedy, J. F. Kennedy, and R. C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[159] E. Kent, S. Hoops, and P. Mendes. Condor-COPASI: high-throughput computing for biochemical networks. *BMC Systems Biology*, 6(1):91, 2012.

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

[161] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

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

[163] V. Klee and G. J. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities III*, pages 159–175. Academic Press, 1972.

[164] P. E. Kloeden and E. Platen. *Numerical solution of stochastic differential equations*, volume 23 of *Stochastic Modelling and Applied Probability*. Springer-Verlag, 1992.

[165] I. Komarov, R. M. D'Souza, and J. Tapia. Accelerating the Gillespie τ-leaping method using Graphics Processing Units. *PLoS ONE*, 7:e37370, 2012.

[166] J. R. Koza, W. Mydlowec, G. Lanza, J. Yu, and M. A. Keane. Reverse engineering of metabolic pathways from observed data using genetic programming. In R. B. Altman et al., editor, *Pacific Symposium on Biocomputing*, volume 6, pages 434–445, 2001.

[167] J. R. Koza, W. Mydlowec, G. Lanza, J. Yu, and M. A. Keane. Automatic computational discovery of chemical reaction networks using genetic programming. *Computational Discovery*, 4660:205–227, 2007.

[168] J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press, 1992.

[169] N. Krasnogor and J. Smith. A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488, 2005.

[170] M. Kubista, J. M. Andrade, M. Bengtsson, A. Forootan, J. Jonák, K. Lind, R. Sindelka, R. Sjöback, B. Sjögreen, L. Strömbom, A. Ståhlberg, and N. Zoric. The real-time polymerase chain reaction. *Molecular Aspects of Medicine*, 27(2–3):95–125, 2006.

[171] T. H. LaBean, H. Yan, J. Kopatsch, F. Liu, E. Winfree, J. H. Reif, and N. C. Seeman. Construction, analysis, ligation, and self-assembly of DNA triple crossover complexes. *Journal of the American Chemical Society*, 122(9):1848–1860, 2000.

[172] P. Larrañaga, C. M. H. Kujipers, R. H. Murga, I. Inza, and S. Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13(2):129–170, 1999.

[173] C. Lavor, L. Liberti, N. Maculan, and A. Mucherino. The discretizable molecular distance geometry problem. *Computational Optimization and Applications*, 52(1):115–146, 2011.

[174] C. Lavor, L. Liberti, A. Mucherino, and N. Maculan. On a discretizable subclass of instances of the molecular distance geometry problem. In D. Shin, editor, *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 804–805. ACM, 2009.

[175] A. Lazcano and S. L. Miller. The origin and early evolution of life: Prebiotic chemistry, the pre-RNA world, and time. *Cell*, 85(6):793 –798, 1996.

[176] P. L'Ecuyer and R. Simard. TestU01: A C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, 33(4), 2007.

[177] P. L'Ecuyer, R. Simard, E. J. Chen, and W. D. Kelton. An object-oriented random-number package with many long streams and substreams. *Operations Research*, 50(6):1073–1075, 2002.

[178] T. Lenser, T. Hinze, B. Ibrahim, and P. Dittrich. Towards evolutionary network reconstruction tools for Systems Biology. In E. Marchiori, J. H. Moore, and J. C. Rajapakse, editors, *Proceedings EvoBIO 2007*, volume 4447 of *Lecture Notes in Computer Science*, pages 132–142. Springer-Verlag, 2007.

[179] M. Levitt. The birth of computational structural biology. *Nature Structural & Molecular Biology*, 8(5):392–393, 2001.

[180] C. Li, M. Donizelli, N. Rodriguez, H. Dharuri, L. Endler, V. Chelliah, L. Li, E. He, A. Henry, M. I. Stefan, J. L. Snoep, M. Hucka, N. Le Novère, and C. Laibe. BioModels Database: An enhanced, curated and annotated resource for published quantitative kinetic models. *BMC Systems Biology*, 4:92, 2010.

[181] C. Li, Q. Ge, M. Nakata, H. Matsuno, and S. Miyano. Modelling and simulation of signal transduction in an apoptosis pathway by using timed Petri nets. *Journal of Biosciences*, 32(1):113–127, 2006.

[182] H. Li and L. R. Petzold. Efficient parallelization of the stochastic simulation algorithm for chemically reacting systems on the Graphics Processing Unit. *International Journal of High Performance Computing Applications*, 24:107–116, 2010.

[183] Y. Li and Z. Xul. An ant colony optimization heuristic for solving maximum independent set problems. In *Proceedings of the Fifth International Conference on Computational Intelligence and Multimedia Applications (ICCIMA'03)*, pages 206–211. IEEE, 2003.

[184] S. Light, P. Kraulis, and A. Elofsson. Preferential attachment in the evolution of metabolic networks. *BMC Genomics*, 6(1):159, 2005.

[185] G. Lillacci and M. Khammash. Parameter estimation and model selection in computational biology. *PLoS Computational Biology*, 6(3):e1000696, 2010.

[186] W. A. Lim. Designing customized cell signalling circuits. *Nature Reviews Molecular Cell Biology*, 11(6):393–403, 2010.

[187] S. C. Lin, W. F. Punch III, and E. D. Goodman. Coarse-grain parallel genetic algorithms: Categorization and new approach. In *Proceedings of Sixth IEEE Symposium on Parallel and Distributed Processing*, pages 28–37. IEEE Computer Society Press, 1994.

[188] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. NVIDIA Tesla: A unified graphics and computing architecture. *IEEE Micro*, 28(2):39–55, 2008.

[189] K. Lipkow, S. S. Andrews, and D. Bray. Simulated diffusion of phosphorylated CheY through the cytoplasm of *Escherichia coli*. *Journal of Bacteriology*, 187:45–53, 2005.

[190] X. Liu and M. Niranjan. State and parameter estimation of the heat shock response system using Kalman and particle filters. *Bioinformatics*, 28(11):1501–1507, 2012.

[191] M. Llorens and J. Oliver. Structural and dynamic changes in concurrent systems: Reconfigurable Petri nets. *IEEE Transactions on Computers*, 53(9):1147–1158, 2004.

[192] J. Logan, K. Edwards, and N. Saunders, editors. *Real-Time PCR: Current Technology and Applications*. Caister Academic Press, 2009.

[193] H. Maamar, A. Raj, and D. Dubnau. Noise in gene expression determines cell fate in *Bacillus subtilis*. *Science*, 317(5837):526–529, 2007.

[194] D. Machado, R. S. Costa, M. Rocha, E. C. Ferreira, B. Tidor, and I. Rocha. Modeling formalisms in systems biology. *AMB Express*, 1(1):1–14, 2011.

[195] Y. Maki, D. Tominaga, M. Okamoto, S. Watanabe, and Y. Eguchi. Development of a system for the inference of large scale genetic networks. In R. B. Altman et al., editor, *Pacific Symposium on Biocomputing*, volume 6, pages 446–458, 2001.

[196] M. Manssen, M. Weigel, and A. K. Hartmann. Random number generators for massively parallel simulations on GPU. *The European Physical Journal-Special Topics*, 210(1):53–71, 2012.

[197] D. Marco, C. Shankland, and D. Cairns. Evolving Bio-PEPA algebra models using Genetic Programming. In T. Soule, editor, *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, pages 177–183. ACM, 2012.

[198] W. R. Mark, R. S. Glanville, K. Akeley, and M. J. Kilgard. CG: A system for programming graphics hardware in a C-like language. *ACM Transactions on Graphics*, 22(3):896–907, 2003.

[199] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):pp. 431–441, 1963.

[200] G. Marsaglia. Xorshift RNGs. *Journal of Statistical Software*, 8(14):1–6, 2003.

[201] I. Martinelli, P. Bucciarelli, and P. Mannucci. Thrombotic risk factors: basic pathophysiology. *Critical Care Medicine*, 38:S3–S9, 2010.

[202] F. J. Massey Jr. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46(253):68–78, 1951.

[203] A. J. Matlin, F. Clark, and C. W. J. Smith. Understanding alternative splicing: towards a cellular code. *Nature Reviews Molecular Cell Biology*, 6(5):386–398, 2005.

[204] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8:3–30, 1998.

[205] H. Matsuno. A new regulatory interactions suggested by simulations for circadian genetic control mechanism in mammals. *Journal of Bioinformatics and Computational Biology*, 4(1):139–153, 2005.

[206] H. Matsuno, Y. Tanaka, H. Aoshima, A. Doi, M. Matsui, and S. Miyano. Biopathways representation and simulation on hybrid functional Petri net. *In Silico Biology*, 3(3):389–404, 2003.

[207] C. Maus, S. Rybacki, and A. M. Uhrmacher. Rule-based multi-level modeling of cell biological systems. *BMC Systems Biology*, 5:166, 2011.

[208] H. H. McAdams and A. Arkin. Gene regulation: Towards a circuit engineering discipline. *Current Biology*, 10(8):318–320, 2000.

[209] H. H. McAdams and L. Shapiro. Circuit simulation of genetic networks. *Science*, 269(5224):650–656, 1995.

[210] O. Medvedik, D. W. Lamming, K. D. Kim, and D. A. Sinclair. MSN2 and MSN4 link calorie restriction and TOR to Sirtuin-mediated lifespan extension in *Saccharomyces cerevisiae*. *PLoS Biology*, 5:2330–2341, 2007.

[211] P. Mendes and D. B. Kell. Non-linear optimization of biochemical pathways: applications to metabolic engineering and parameter estimation. *Bioinformatics*, 14:869–883, 1998.

[212] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[213] A. D. Michelson. *Platelets*. Elsevier, 2nd edition, 2007.

[214] J. Miller and P. Thomson. Cartesian genetic programming. In R. Poli et al., editor, *Proceedings of the Third European Conference on Genetic Programming*, volume 1802 of *Lecture Notes in Computer Science*, pages 121–132. Springer-Verlag, 2000.

[215] R. Milner. *Communicating and Mobile Systems: The π-Calculus*. Cambridge University Press, 1999.

[216] C. G. Moles, P. Mendes, and J. R. Banga. Parameter estimation in biochemical pathways: a comparison of global optimization methods. *Genome Research*, 13(11):2467–2474, 2003.

[217] J. Monod and F. Jacob. General conclusions: teleonomic mechanisms in cellular metabolism, growth, and differentiation. In *Cold Spring Harbor Symposia on Quantitative Biology*, volume 26, pages 389–401. Cold Spring Harbor Laboratory Press, 1961.

[218] O. Morandin and E. Kato. Virtual Petri nets as a modular modeling method for planning and control tasks of FMS. *International Journal of Computer Integrated Manufacturing*, 18(2-3):100–106, 2005.

[219] J. J. Moré and Z. Wu. Global continuation for distance geometry problems. *SIAM Journal on Optimization*, 7(3):814–836, 1997.

[220] J. J. Moré and Z. Wu. Distance geometry optimization for protein structures. *Journal of Global Optimization*, 15(3):219–234, 1999.

[221] The PyMOL Web Page. http://www.pymol.org.

[222] B. Munsky and M. Khammash. The finite state projection algorithm for the solution of the Chemical Master Equation. *The Journal of Chemical Physics*, 124:044104, 2006.

[223] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[224] K. Nakamura, R. Yoshida, M. Nagasaki, S. Miyano, and T. Higuchi. Parameter estimation of *in silico* biological pathways with particle filtering towards a petascale computing. In *Pacific Symposium on Biocomputing*, volume 14, pages 227–238, 2009.

[225] N. Nandapalan, R. P. Brent, L. M Murray, and A. P. Rendell. High-performance pseudo-random number generation on Graphics Processing Units. In R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, editors, *Parallel Processing and Applied Mathematics*, Lecture Notes in Computer Science, pages 609–618. Springer-Verlag, 2012.

[226] D. L. Nelson and M. M. Cox. *Lehninger Principles of Biochemistry.* W. H. Freeman & Company, 4th edition, 2004.

[227] J. Nickolls and W. J. Dally. The GPU computing era. *IEEE Micro*, 30(2):56–69, 2010.

[228] M. S. Nobile. Evolutionary inference of biochemical interaction networks accelerated on graphics processing units. In *Proceedings of the 11th International Conference on High Performance Computing & Simulation 2013 (HPCS 2013)*, pages 668–670, 2013.

[229] M. S. Nobile, D. Besozzi, P. Cazzaniga, and G. Mauri. The foundation of evolutionary Petri nets. In G. Balbo and M. Heiner, editors, *Proceedings of the 4th International Workshop on Biological Processes & Petri Nets (BioPPN 2013)*, volume 988, pages 60–74. CEUR Workshop Proceedings, 2013.

[230] M. S. Nobile, D. Besozzi, P. Cazzaniga, and G. Mauri. GPU-accelerated simulations of mass-action kinetics models with cupSODA. *The Journal of Supercomputing*, 69(1):17–24, 2014.

[231] M. S. Nobile, D. Besozzi, P. Cazzaniga, G. Mauri, and D. Pescini. Estimating reaction constants in stochastic biological systems with a multi-swarm PSO running on GPUs. In T. Soule, editor, *Proceedings of the fourteenth International Conference on Genetic and Evolutionary Computation Conference Companion*, GECCO Companion '12, pages 1421–1422. ACM, 2012.

[232] M. S. Nobile, D. Besozzi, P. Cazzaniga, G. Mauri, and D. Pescini. A GPU-based multi-swarm PSO method for parameter estimation in stochastic biological systems exploiting discrete-time target series. In M. Giacobini, L. Vanneschi, and W. Bush, editors, *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics. 10th European Conference, EvoBIO 2012. Proceedings*, volume 7246 of *Lecture Notes in Computer Science*, pages 74–85. Springer-Verlag, 2012.

[233] M. S. Nobile, D. Besozzi, P. Cazzaniga, G. Mauri, and D. Pescini. cupSODA: a CUDA-powered simulator of mass-action kinetics. In V. Malyshkin, editor, *Proceedings of 12th International Conference on Parallel Computing Technologies (PaCT 2013)*, volume 7979 of *Lecture Notes in Computer Science*, pages 344–357. Springer-Verlag, 2013.

[234] M. S. Nobile, D. Besozzi, P. Cazzaniga, D. Pescini, and G. Mauri. Reverse engineering of kinetic reaction networks by means of Cartesian Genetic Programming and Particle Swarm Optimization. In *2013 IEEE Congress on Evolutionary Computation*, volume 1, pages 1594–1601. IEEE, 2013.

[235] M. S. Nobile, P. Cazzaniga, D. Besozzi, D. Pescini, and G. Mauri. cuTauLeaping: A GPU-powered tau-leaping stochastic simulator for massive parallel analyses of biological systems. *PLoS ONE*, 9(3):e91963, 2014.

[236] M. S. Nobile, D. Cipolla, P. Cazzaniga, and D. Besozzi. GPU-powered evolutionary design of mass-action based models of gene regulation. In H. Iba

and N. Noman, editors, *Evolutionary Algorithms in Gene Regulatory Network Research*. John Wiley & Sons, 2014. In press.

[237] M. S. Nobile, A. G. Citrolo, P. Cazzaniga, D. Besozzi, and G. Mauri. A memetic hybrid method for the molecular distance geometry problem with incomplete information. In *Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC2014)*, pages 1014–1021, 2014.

[238] M.S. Nobile, P. Cazzaniga, D. Besozzi, D. Cipolla, and G. Mauri. Parameter estimation on graphics processing units: a multi-swarm approach for stochastic cellular systems. *IEEE Transactions on Evolutionary Computation*, 2014. Submitted.

[239] N. Noman and H. Iba. Inference of gene regulatory networks using S-system and differential evolution. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, pages 439–446. ACM, 2005.

[240] J. Norberg and L. Nilsson. Advances in biomolecular simulations: methodology and recent applications. *Quarterly Reviews of Biophysics*, 36(03):257–306, 2003.

[241] J. Nummela and J. A. Bryant. Evolving Petri nets to represent metabolic pathways. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, pages 2133–2139. ACM, 2005.

[242] Nvidia. CUDA Toolkit 5.0 CURAND Guide, March 2012.

[243] Nvidia. Dynamic parallelism in CUDA, 2012.

[244] Nvidia. Nvidia CUDA C Programming Guide, 2012.

[245] Nvidia. Nvidia's Next Generation CUDA Compute Architecture: Kepler GK110, 2012.

[246] Nvidia. CUDA Toolkit 6.0 CURAND Guide, February 2014.

[247] Nvidia. Nvidia CUDA C Programming Guide v6.5, 2014.

[248] J. D. Orth, I. Thiele, and B. Ø. Palsson. What is flux balance analysis? *Nature Biotechnology*, 28(3):245–8, 2010.

[249] J. A. Papin, T. Hunter, B. Ø. Palsson, and S. Subramaniam. Reconstruction of cellular signalling networks and analysis of their properties. *Nature Reviews Molecular Cell Biology*, 9(2):99–111, 2005.

[250] G. Pasquale, C. Maj, A. Clematis, E. Mosca, L. Milanesi, I. Merelli, and D. D'Agostino. A CUDA implementation of the Spatial TAU-leaping in Crowded Compartments (STAUCC) simulator. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pages 609–616. IEEE, 2014.

[251] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1st edition, 1988.

[252] F. Pereira, P. Machado, E. Costa, and A. Cardoso. Graph based crossover - a case study with the busy beaver problem. In W. Banzhaf et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999)*, volume 2, pages 1149–1155. Morgan Kaufmann, 1999.

[253] D. Peri and F. Tinti. A multistart gradient-based algorithm with surrogate model for global optimization. *Communications in Applied and Industrial Mathematics*, 3(1):1–22, 2012.

[254] B. E. Perrin, L. Ralaivola, A. Mazurie, S. Bottani, J. Mallet, and F. d'Alche Buc. Gene networks inference using dynamic bayesian networks. *Bioinformatics*, 19(2):138–148, 2003.

[255] D. Pescini, P. Cazzaniga, D. Besozzi, G. Mauri, L. Amigoni, S. Colombo, and E. Martegani. Simulation of the Ras/cAMP/PKA pathway in budding yeast highlights the establishment of stable oscillatory states. *Biotechnology Advances*, 30:99–107, 2012.

[256] C. A. Petri. *Kommunikation mit automata.* PhD thesis, Ph.D. thesis, University of Bonn, Bonn, Germany, 1962.

[257] L. Petzold. Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations. *SIAM Journal of Scientific and Statistical Computing*, 4:136–148, 1983.

[258] L. Piela, J. Kostrowicki, and H. A. Scheraga. On the multiple-minima problem in the conformational analysis of molecules: deformation of the potential energy hypersurface by the diffusion equation method. *The Journal of Physical Chemistry*, 93(8):3339–3346, 1989.

[259] Pixar, Inc. RenderMan Interface Specification, Version 3.0, 1988.

[260] R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1(1):33–57, 2007.

[261] R. Poli and W. B. Langdon. Schema theory for genetic programming with one-point crossover and point mutation. *Evolutionary Computation*, 6(3):231–252, 1998.

[262] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza. *A Field Guide to Genetic Programming.* Lulu Enterprises, UK, 2008.

[263] J. R. Pomerening. Uncovering mechanisms of bistability in biological systems. *Current Opinion in Biotechnology*, 19(4):381–388, 2008.

[264] S. K. Poovathingal and R. Gunawan. Global parameter estimation methods for stochastic biochemical systems. *BMC Bioinformatics*, 11(414), 2010.

[265] K. R. Popper. *Logik der Forschung*, volume 4. JCB Mohr (Paul Siebeck), 1982.

[266] K. R. Popper. *Conjectures and Refutations: The Growth of Scientific Knowledge.* Routledge, 2014.

[267] N. D. Price, J. A. Papin, and B. Ø. Palsson. Determination of redundancy and systems properties of the metabolic network of *Helicobacter pylori* using genome-scale extreme pathway analysis. *Genome Research*, 12(5):760–769, 2002.

[268] S. Pricl, M. Ferrone, M. Fermeglia, F. Amato, C. Cosentino, M. M.-C. Cheng, R. Walczak, and M. Ferrari. Multiscale modeling of protein transport in silicon membrane nanochannels. Part 1. Derivation of molecular parameters from computer simulations. *Biomedical Microdevices*, 8(4):277–290, 2006.

[269] RCSB Protein Data Bank. www.rcsb.org.

[270] S. Pulikanti and A. Singh. An artificial bee colony algorithm for the quadratic knapsack problem. In C.-S. Leung, M. Lee, and J. H. Chan, editors, *16th International Conference on Neural Information Processing*, volume 5864 of *Lecture Notes on Computer Science*, pages 196–205. Springer-Verlag, 2009.

[271] A. Raj and A. van Oudenaarden. Nature, nurture, or chance: stochastic gene expression and its consequences. *Cell*, 135(2):216–226, 2008.

[272] R. Ramakrishna, J. S. Edwards, A. McCulloch, and B. Ø. Palsson. Flux-balance analysis of mitochondrial energy metabolism: consequences of systemic stoichiometric constraints. *American Journal of Physiology-Regulatory, Integrative and Comparative Physiology*, 280(3):R695–R704, 2001.

[273] A. Raue, V. Becker, U. Klingmüller, and J. Timmer. Identifiability and observability analysis for experimental design in nonlinear dynamical models. *Chaos*, 20(4):045105, 2010.

[274] I. Rechenberg. *Evolution Strategy: Optimization of Technical systems by means of biological evolution*. Fromman-Holzboog, 1973.

[275] A. Regev, W. Silverman, and E. Shapiro. Representation and simulation of biochemical processes using the π-calculus process algebra. In R. B. Altman et al., editor, *Pacific Symposium on Biocomputing*, volume 6, pages 459–470, 2001.

[276] S. Reinker, R. M. Altman, and J. Timmer. Parameter estimation in stochastic biochemical reactions. *IEE Proceedings - Systems Biology*, 153:168–178, 2006.

[277] J. Reiterman, V. Rödl, and E. Šinajová. Geometrical embeddings of graphs. *Discrete Mathematics*, 74(3):291–319, 1989.

[278] T. Renné, A. H. Schmaier, K. F. Nickel, M. Blombäck, and C. Maas. *In vivo* roles of factor XII. *Blood*, 120(22):4296–4303, 2012.

[279] P. Richmond, S. Coakley, and D. M. Romano. A high performance agent based modelling framework on graphics card hardware with CUDA. In K. S. Decker, J. S. Sichman, and C. Castelfranchi, editors, *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, volume 2, pages 1125–1126. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[280] P. Richmond, D. Walker, S. Coakley, and D. Romano. High performance cellular level agent-based simulation with FLAME for the GPU. *Briefings in Bioinformatics*, 11(3):334–347, 2010.

[281] E. Roberts, J. E. Stone, L. Sepúlveda, W.-M. W. Hwu, and Z. Luthey-Schulten. Long time-scale simulations of *in vivo* diffusion using GPU hardware. In *23rd IEEE International Parallel & Distributed Processing Symposium*, pages 1–8. IEEE, 2009.

[282] A. M. Robinson and D. H. Williamson. Physiological roles of ketone bodies as substrates and signals in mammalian tissues. *Physiological Reviews*, 60(1):143–187, 1980.

[283] K. Rogers. *Blood: Physiology and Circulation*. The Human Body. Britannica Educational Publishing, 2010.

[284] J. Romero and C. Cotta. Optimization by island-structured decentralized particle swarms. In B. Reusch, editor, *Computational Intelligence, Theory and Applications*, volume 33 of *Advances in Soft Computing*, pages 25–33. Springer, 2005.

[285] B. J. Ross. The evolution of higher-level biochemical reaction models. *Genetic Programming and Evolvable Machines*, 13(1):3–31, 2012.

[286] L. Roth and B. Asimow. The rigidity of graphs. *Transactions of the American Mathematical Society*, 245:279–289, 1978.

[287] P. W. K. Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, 2006.

[288] P. Rumschinski, S. Borchers, S. Bosio, R.Weismantel, and R. Findeisen. Set-base dynamical parameter estimation and model invalidation for biochemical reaction networks. *BMC Systems Biology*, 4(69), 2010.

[289] C. Ryan and M. Keijzer. An analysis of diversity of constants of genetic programming. In C. Ryan et al., editor, *Genetic Programming*, pages 404–413. Springer-Verlag, 2003.

[290] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W. W. Hwu. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '08, pages 73–82. ACM, 2008.

[291] H. Salis and Y. Kaznessis. Accurate hybrid stochastic simulation of a system of coupled chemical or biochemical reactions. *The Journal of Chemical Physics*, 122(5):054103, 2005.

[292] H. Salis, V. Sotiropoulos, and Y. N. Kaznessis. Multiscale Hy3S: Hybrid stochastic simulation for supercomputers. *BMC Bioinformatics*, 7(1):93, 2006.

[293] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola. *Global Sensitivity Analysis: The Primer.* Wiley-Interscience, 2008.

[294] G. M. Santangelo. Glucose signaling in *Saccharomyces cerevisiae. Microbiology and Molecular Biology Reviews*, 70(1):253–282, 2006.

[295] M. Santillán. On the use of the Hill functions in mathematical models of gene regulatory networks. *Mathematical Modelling of Natural Phenomena*, 3(2):85–97, 2008.

[296] M. A. Savageau. Comparison of classical and autogenous systems of regulation in inducible operons. *Nature*, 252:546–549, 1974.

[297] M. A. Savageau. *Biochemical Systems Analysis. A Study of Function and Design in Molecular Biology.* Addison-Wesley, Reading, Massachusetts, USA, 1976.

[298] H. Sawai and S. Kizu. Parameter-free genetic algorithm inspired by "disparity theory of evolution". In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, pages 702–711. Springer-Verlag, 1998.

[299] J. B. Saxe. Embeddability of weighted graphs in k-space is strongly NP-hard. In *Proc. of 17th Allerton Conference in Communications, Control and Computing*, pages 480–489, 1979.

[300] M. C. Schatz, C. Trapnell, A. L. Delcher, and A. Varshney. High-throughput sequence alignment using Graphics Processing Units. *BMC bioinformatics*, 8(1):474, 2007.

[301] C. H. Schilling, D. Letscher, and B. Ø. Palsson. Theory for the systemic definition of metabolic pathways and their use in interpreting metabolic function from a pathway-oriented perspective. *Journal of Theoretical Biology*, 203(3):229–248, 2000.

[302] T. Schlick. *Molecular Modeling and Simulation: An Interdisciplinary Guide*, volume 21 of *Interdisciplinary Applied Mathematics.* Springer-Verlag, 2nd edition, 2010.

[303] T. Schreiber. Measuring information transfer. *Physical Review Letters*, 85(2):461, 2000.

[304] H.-P. P. Schwefel. *Evolution and Optimum Seeking: The Sixth Generation.* John Wiley & Sons, 1993.

[305] T. D. Seeley. The honey bee colony as a superorganism. *American Scientist*, 77(6):546–553, 1989.

[306] U. Seligsohn and A. Lubetsky. Genetic susceptibility to venous thrombosis. *The New England Journal of Medicine*, 344(16):1222–1231, 2001.

[307] V. Shahrezaei and P. S. Swain. The stochastic nature of biochemical networks. *Current Opinion in Biotechnology*, 19(4):369–374, 2008.

[308] Y. Shi and R. C. Eberhart. Parameter selection in particle swarm optimization. In V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, editors, *Evolutionary Programming VII*, volume 1447 of *Lecture Notes in Computer Science*, pages 591–600. Springer-Verlag, 1998.

[309] Y. Shi and R. C. Eberhart. Fuzzy adaptive particle swarm optimization. In *Proceedings of the 2001 Congress on Evolutionary Computation*, volume 1, pages 101–106, Seoul, South Korea, May 2001. IEEE.

[310] T. S. Shimizu, S. V. Aksenov, and D. Bray. A spatially extended stochastic model of the bacterial chemotaxis signalling pathway. *Journal of Molecular Biology*, 329(2):291–309, 2003.

[311] E. Shockley and C. F. Lopez. Towards understanding cellular proliferation and death signal processing in cancer. In *Proceedings of the Eighth Q-bio Conference*, 2014.

[312] R. Sidje, K. Burrage, and S. MacNamara. Inexact uniformization method for computing transient distributions of Markov chains. *SIAM Journal on Scientific Computing*, 29:2562–2580, 2007.

[313] J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. M. Jones, and I. Birol. ABySS: a parallel assembler for short read sequence data. *Genome Research*, 19(6):1117–1123, 2009.

[314] A. Sit and Z. Wu. Solving a generalized distance geometry problem for protein structure determination. *Bulletin of Mathematical Biology*, 73(12):2809–2836, 2011.

[315] A. Sit, Z. Wu, and Y. Yuan. A geometric buildup algorithm for the solution of the distance geometry problem using least-squares approximation. *Bulletin of Mathematical Biology*, 71(8):1914–33, 2009.

[316] W. Situ and P. Tsang. A parameter estimation method for biological systems modeled by ODEs/DDEs models using spline approximation and differential evolution algorithm. *IEEE Transactions on Computational Biology and Bioinformatics*, 99, 2014. PrePrints.

[317] W. W. Soon, M. Hariharan, and M. P. Snyder. High-throughput sequencing for biology and medicine. *Molecular systems biology*, 9(640), 2013.

[318] T. T. Soong. *Random Differential Equations in Science and Engineering.* Academic Press, 1973.

[319] M. Stein, R. R. Gabdoulline, and R. C. Wade. Bridging from molecular simulation to biochemical networks. *Current Opinion in Structural Biology*, 17(2):166–172, 2007.

[320] J. Stelling. Mathematical models in microbial systems biology. *Current Opinion in Microbiology*, 7(5):513–518, 2004.

[321] R. Steuer, C. Zhou, and J. Kurths. Constructive effects of fluctuations in genetic and biochemical regulatory systems. *BioSystems*, 72(3):241–251, 2003.

[322] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton University Press, 2010.

[323] J. E. Stone, D. J. Hardy, I. S. Ufimtsev, and K. Schulten. GPU-accelerated molecular modeling coming of age. *Journal of Molecular Graphics and Modelling*, 29(2):116–125, 2010.

[324] R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[325] T. Stützle and M. Dorigo. ACO algorithms for the traveling salesman problem. In K. Miettinen, P. Neittaanmäki, M. M. Mäkelä, and J. Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science*, pages 163–183. Wiley, 1999.

[326] M. Sugimoto, S. Kikuchi, and M. Tomita. Reverse engineering of biochemical equations from time-course data by means of genetic programming. *BioSystems*, 80(2):155–164, 2005.

[327] G. Szederkenyi, J. R. Banga, and A. A. Alonso. Inference of complex biological networks: distinguishability issues and optimization-based solutions. *BMC Systems Biology*, 5(1):177, 2011.

[328] R. Tanese. Distributed genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 434–439. Morgan Kaufmann, 1989.

[329] D. C. Teller, T. Okada, C. A. Behnke, K. Palczewski, and R. E. Stenkamp. Advances in determination of a high-resolution three-dimensional structure of rhodopsin, a model of G-protein-coupled receptors (GPCRs). *Biochemistry*, 40(26):7761–7772, 2001.

[330] G. Terrazas, M. Gheorghe, G. Kendall, and N. Krasnogor. Evolving tiles for automated self-assembly design. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 2001–2008. IEEE, 2007.

[331] G. Theraulaz and E. Bonabeau. A brief history of stigmergy. *Artificial Life*, 5(2):97–116, 1999.

[332] T. Tian and K. Burrage. Binomial leap methods for simulating stochastic chemical kinetics. *The Journal of Chemical Physics*, 121(21):10356–10364, 2004.

[333] M. Tomassini. Parallel and distributed evolutionary algorithms: a review. In K. Miettinen, M. Makela, P. Neittanmaki, and J. Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science*, pages 113–133. J. Wiley, New York, 1999.

[334] M. Tomassini, L. Vanneschi, J. Cuendet, and F. Fernández. A new technique for dynamic size populations in genetic programming. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 1, pages 486–493. IEEE, 2004.

[335] C. T. Trinh, A. Wlaschin, and F. Srienc. Elementary mode analysis: a useful metabolic pathway analysis tool for characterizing cellular metabolism. *Applied Microbiology and Biotechnology*, 81(5):813–826, 2009.

[336] K.-Y. Tsai and F.-S. Wang. Evolutionary optimization with data collocation for reverse engineering of biological networks. *Bioinformatics*, 21(7):1180–1188, 2005.

[337] Y. Del Valle, G.K. Venayagamoorthy, S. Mohagheghi, J.C. Hernandez, and R.G. Harley. Particle swarm optimization: Basic concepts, variants and applications in power systems. *IEEE Transactions on Evolutionary Computation*, 12(2):171–195, 2008.

[338] V. A. van Hylckama, I. K. van der Linden, R. M. Bertina, and F. R. Rosendaal. High levels of factor IX increase the risk of venous thrombosis. *Blood*, 95:3678–3682, 2000.

[339] N. G. van Kampen. *Stochastic Processes in Physics and Chemistry.* Elsevier, Amsterdam, The Netherlands, 3rd edition, 2001.

[340] W. Vance, A. Arkin, and J. Ross. Determination of causal connectivities of species in reaction networks. *Proceedings of the National Academy of Sciences*, 99(9):5816–5821, 2002.

[341] L. Vanneschi, D. Codecasa, and G. Mauri. A comparative study of four parallel and distributed PSO methods. *New Generation Computing*, 29(2):129–161, 2011.

[342] M. Vellela and H. Qian. Stochastic dynamics and non-equilibrium thermodynamics of a bistable chemical system: the Schlögl model revisited. *Journal of the Royal Society Interface*, 6(39):925–940, 2009.

[343] D. Voet and J. G. Voet. *Biochemistry.* John Wiley & Sons, 4th edition, 2011.

[344] E. O. Voit. *Computational Analysis of Biochemical Systems.* Cambridge University Press, Cambridge, UK, 2000.

[345] K. Voss, M. Heiner, and I. Koch. Steady state analysis of metabolic pathways using Petri nets. *In Silico Biology*, 3(3):367–387, 2003.

[346] C. E. Walsh and K. M. Batt. Hemophilia clinical gene therapy: brief review. *Translational Research*, 161:307–312, 2013.

[347] H. Wang, L. Qian, and E. Dougherty. Inference of gene regulatory networks using S-system: a unified approach. *IET Systems Biology*, 4(2):145–156, 2010.

[348] L. Wang, G. Renault, H. Garreau, and M. Jacquet. Stress induces depletion of Cdc25p and decreases the cAMP producing capability in *Saccharomyces cerevisiae. Microbiology*, 150(10):3383–91, 2004.

[349] Y. Wang, S. Christley, E. Mjolsness, and X. Xie. Parameter inference for discretely observed stochastic kinetic models using stochastic gradient descent. *BMC Systems Biology*, 4(1):99, 2010.

[350] J. R. Weimar and J.-P. Boon. Class of cellular automata for reaction-diffusion systems. *Physical Review E*, 49(2):1749, 1994.

[351] J. N. Weiss. The Hill equation revisited: uses and misuses. *The FASEB Journal*, 11(11):835–841, 1997.

[352] W. H. Wen-mei. *GPU Computing Gems Jade Edition.* Morgan Kaufmann, 2011.

[353] S. Widder, J. Macía, and R. Solé. Monomeric bistability and the role of autoloops in gene regulation. *PLoS ONE*, 4(4):e5399, 2009.

[354] T. Wilhelm. The smallest chemical reaction system with bistability. *BMC Systems Biology*, 3(1):90, 2009.

[355] D. Wilkinson. Stochastic modelling for quantitative description of heterogeneous biological systems. *Nature Reviews Genetics*, 10(2):122–133, 2009.

[356] D. S. Wishart, R. Yang, D. Arndt, P. Tang, and J. Cruz. Dynamic cellular automata: an alternative approach to cellular simulation. *In Silico Biology*, 5(2):139–161, 2005.

[357] V. Wolf, R. Goel, M. Mateescu, and T. A. Henzinger. Solving the Chemical Master Equation using sliding windows. *BMC Systems Biology*, 4(1):42, 2010.

[358] O. Wolkenhauer, M. Ullah, W. Kolch, and C. Kwang-Hyun. Modeling and simulation of intracellular dynamics: choosing an appropriate framework. *IEEE Transactions on Nanobiosciences*, 3(3):200–7, 2004.

[359] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[360] K. Wüthrich. NMR studies of structure and function of biological macromolecules (Nobel lecture). *Angewandte Chemie International Edition*, 42(29):3340–63, 2003.

[361] S. Xu and Y. Rahmat-Samii. Boundary conditions in Particle Swarm Optimization revisited. *IEEE Transactions on Antennas and Propagation*, 55(3):760–765, 2007.

[362] J. Yang, M. I. Monine, J. R. Faeder, and W. S. Hlavacek. Kinetic Monte Carlo method for rule-based modeling of biochemical networks. *Physical Review E*, 78(3):031910, 2008.

[363] F. Zafari, G. M. Khan, M. Rehman, and S. Ali Mahmud. Evolving recurrent neural network using cartesian genetic programming to predict the trend in foreign currency exchange rates. *Applied Artificial Intelligence*, 28(6):597–628, 2014.

[364] S. Zaman, S. I. Lippman, L. Schneper, N. Slonim, and J. R. Broach. Glucose regulates transcription in yeast through a network of signaling pathways. *Molecular Systems Biology*, 5(1):1–14, 2009.

[365] D. R. Zerbino and E. Birney. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, 18(5):821–829, 2008.

[366] I. Zevedei-Oancea and S. Schuster. Topological analysis of metabolic networks based on Petri net theory. *In Silico Biology*, 3(3):0029, 2003.

[367] B.-T. Zhang and H. Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38, 1995.

[368] J. Zhang, L. T. Watson, and Y. Cao. A modified uniformization method for the solution of the Chemical Master Equation. Technical Report TR-07-31, Computer Science, Virginia Tech, 2007.

[369] K. Zhang, J. Yi, J. Liu, and W. Hu. Multiobjective genetic algorithm for optimized DNA sequences for DNA self-assembly. In L. Pan, G. Păun, M. J. Pérez-Jiménez, and T. Song, editors, *Bio-Inspired Computing – Theories and Applications. 9th International Conference, BIC-TA 2014, Wuhan, China, October 16-19, 2014. Proceedings*, pages 591–597. Springer-Verlag, 2014.

[370] Y. Zhou, J. Liepe, X. Sheng, M. P. H. Stumpf, and C. Barnes. GPU accelerated biochemical network simulation. *Bioinformatics*, 27(6):874–876, 2011.

Part III

Appendix

Appendix A

Reaction-based models of biological systems

A.1 Michaelis-Menten kinetics

The Michaelis-Menten (MM) model is a simple example of enzyme kinetics consisting in 3 reactions, which describe the catalytic transformation of a substrate S into a product P mediated by the activitity of an enzyme E, passing through the reversible formation of an enzyme-substrate intermediate complex ES [226]. This enzymatic kinetic is the basis of most biochemical processes occurring in living cells [226]. These reactions, together with the values of the associated stochastic constants, are given in Table A.1. The initial molecular amounts used as default in this work, given as number of molecules, are listed in Table A.2.

Table A.1: Michaelis-Menten model.

No.	*Reactants*	*Products*	*Stochastic constant*
R_1	$S + E$	ES	0.0025
R_2	ES	$S + E$	0.1
R_3	ES	$E + P$	5.0

Table A.2: Initial molecular amounts of the Michaelis-Menten model.

Molecular species	*Initial amount*
S	1000
E	750
ES	0
P	0

A.2 Prokaryotic auto-regulatory gene network

The prokaryotic gene network (PGN) [349] describes an auto-regulation mechanism of gene expression, whereby a gene (DNA) that codes for a protein (P) is inhibited by binding to a dimer of the protein itself ($DNA{:}P_2$). Gene expression is a good example of stochasticity in biological systems: the transcriptional regulators are present in a few copies, so that the binding and release of the regulators can be expressed in probabilistic terms.

The reactions describing the molecular interactions occurring in PGN, together with the values of the associated stochastic constants, are given in Table A.3. The default initial molecular amount used in this thesis is DNA=200 molecules; all other molecular species are generated in the system as long as reactions are applied.

Table A.3: Prokaryotic auto-regulatory gene network model.

No.	*Reactants*	*Products*	*Stochastic constant*
R_1	$DNA + P_2$	$DNA{:}P_2$	0.1
R_2	$DNA{:}P_2$	$DNA + P_2$	0.7
R_3	DNA	$DNA + mRNA$	0.35
R_4	$mRNA$	$\emptyset$	0.3
R_5	$2P$	P_2	0.1
R_6	P_2	$2P$	0.9
R_7	$mRNA$	$mRNA + P$	0.2
R_8	P	$\emptyset$	0.1

Remark: $\emptyset$ denotes the degradation of the reactant

A.3 The Schlögl system

The Schlögl system [342, 354] is one of the simplest prototypes of chemical systems presenting a bistable dynamical behavior, i.e., the capacity of switching between two different stable steady states in response to some chemical signaling (see, e.g.,[64, 263, 353] and references therein). The Schlögl model consists of 4 chemical reactions and 3 molecular species, listed in Table A.4. The initial molecular amounts used in this thesis are given in Table A.5.

Table A.4: The Schlögl model.

No.	*Reactants*	*Products*	*Stochastic constant*
R_1	$A + 2X$	$3X$	$3 \cdot 10^{-7}$
R_2	$3X$	$A + 2X$	$1 \cdot 10^{-4}$
R_3	B	X	$1 \cdot 10^{-3}$
R_4	X	B	3.5

Table A.5: Initial molecular amounts of the Schlögl model.

Molecular species	*Initial amount*
A	$^*1 \cdot 10^5$
B	$^*2 \cdot 10^5$
X	250

*The amounts of species A, B are kept constant during the execution of simulations. All molecular amounts are expressed as number of molecules.

A.4 Blood Coagulation Cascade model

The BCC model considered by the coagSODA simulator (Section 5.2) is a slightly reduced version of the "Platelet-Plasma" deterministic model defined in [56], built upon a previous model [136], which describes all parts of blood coagulation: the platelets activation and aggregation; the extrinsic, intrinsic and common pathways (with the exception of factor XIII); the action of several inhibitory molecules (Tissue Factor Pathway Inhibitor, antithrombin III, C1-inhibitor, α1-antitrypsin and α2-antiplasmin).

Since the aim of coagSODA is the investigation of blood coagulation *in vivo*, a small set of reactions given in [56] that have no effect on the clotting time was excluded. To be more precise, the model presented here does not consider the reactions occurring *in vitro* (namely, entries 28 and 35 in Table 1 in [56]), as well as the reactions downstream the fibrinogen conversion, that is, the interactions between the fibrin polymers, thrombin and antithrombin III (namely, entries 55, 56 and 57 in Table 1 in [56]).

For the sake of completeness, Table A.6 describes the BCC model considered by coagSODA, overall consisting in 96 reactions among 71 molecular species.

The model can be partitioned into four functional modules:

1. The first module corresponds to the *extrinsic pathway*, which consists in:
   - the formation of a complex between Tissue Factor and factor VII (modeled by reactions $R_1, \dots, R_4$);
   - the activation of factor VII by the complex between Tissue Factor and factor VIIa (reaction R_5);
   - the activation of factor IX by the complex between Tissue Factor and factor VIIa (reactions R_{13}, R_{14}, R_{15}), and by factor VIIa (reactions R_{78}, R_{79}, R_{80});
   - the activation of factor X by the complex between Tissue Factor and factor VIIa ($R_8, \dots, R_{12}$), and by factor VIIa (R_{81}, R_{82}, R_{83}).
2. The second module corresponds to the *intrinsic pathway*, which consists in:
   - the formation of a complex between factor VIIIa and factor IXa (modeled by reactions R_{18} and R_{19});
   - the activation of factor X by the complex between factor VIIIa and factor IXa (reactions R_{20}, R_{21}, R_{22}), and by factor IXa (reactions R_{72}, R_{73}, R_{74});
   - the activation of factor XII by factor XII itself (reaction R_{44}), by factor XIIa (reactions R_{45}, R_{46}, R_{47}) and by kallikrein (reactions R_{51}, R_{52}, R_{53});

- the activation of prekallikrein by factor XIIa (reactions R_{48}, R_{49}, R_{50}) and by kallikrein (reaction R_{54});
- the activation of factor XI by factor XIIa (reactions R_{61}, R_{62}, R_{63}) and by factor XIa (reaction R_{64});
- the activation of factor IX by factor XIa (reactions R_{69}, R_{70}, R_{71});
- the dissociation of free factor VIIIa (reactions R_{23} and R_{24});
- the dissociation of factor VIIIa in complex with other factors (reactions R_{25} and R_{26}).

3. The third module corresponds to the *common pathway*, which consists in:

   - the activation of factor II by factor Xa (modeled by reaction R_{16}) and by the complex between factor Xa and factor Va, through the formation of the intermediate meizothrombin (reactions $R_{30}, \ldots, R_{33}$);
   - the activation of factor VII by factor Xa (reaction R_6) and by factor IIa (reaction R_7);
   - the activation of factor VIII by factor Xa (reactions R_{75}, R_{76}, R_{77}) and by factor IIa (reaction R_{17});
   - the formation of a complex between factor Xa and factor Va (reactions R_{28} and R_{29});
   - the activation of fibrinogen by factor IIa (reactions $R_{84}, \ldots, R_{96}$);
   - the activation of factor V by factor IIa (reaction R_{27});
   - the activation of factor XI by factor IIa (reactions R_{58}, R_{59}, R_{60}).

4. The fourth module describes the *inhibition of coagulation*, carried out through the main inhibitors of the BCC (antrithrombin, Tissue Factor Pathway Inhibitor, C1-inihibitor, α1-antitrypsin and α2-antiplasmin). These are tight binding inhibitors belonging to the serpin superfamily, which form irreversible complexes. This module consists in:

   - the inhibition of factor Xa by Tissue Factor Pathway Inhibitor (modeled by reactions R_{34} and R_{35});
   - the inhibition of the complex between Tissue Factor, factor VIIa and factor Xa by Tissue Factor Pathway Inhibitor (reactions R_{36}, R_{37}, R_{38});
   - the inhibition of factor IIa by antithrombin (reactions R_{40} and R_{42});

- the inhibition of factor Xa by antithrombin (reaction R_{39});
- the inhibition of factor IXa by antithrombin (reaction R_{41});
- the inhibition of factor XIa by antithrombin (reaction R_{65}) and by C1-inhibitor (reaction R_{66});
- the inhibition of factor XIIa by C1-inhibitor (reaction R_{56}), by antithrombin (reaction R_{57}), by α1-antitrypsin (reaction R_{67}) and by α2-antiplasmin (reaction R_{68});
- the inhibition of the complex between Tissue Factor and factor VIIa by antithrombin (reaction R_{43});
- the inhibition of kallikrein (reaction R_{55}).

The values of the initial concentrations of the molecular species occurring in the BCC model are given in Table A.7. According to [56], the concentration of complexes and active factors were set to 0, except for the active factor VII, which is physiologically present in the blood circulation, even in the absence of a damage, in a concentration that is approximately equal to 1% of the corresponding inactive factor [117].

The system of ODEs, needed to carry out the simulations and the PSA presented in Section 5.2.3, was derived from the reactions given in Table A.6 according to the *mass-action law*, with the exception of 14 reactions belonging to the set $S_\varepsilon = \{R_{19}, R_{21}, R_{23}, R_{29}, R_{31}, R_{35}, R_{46}, R_{49}, R_{52}, R_{62}, R_{70}, R_{73}, R_{76}, R_{82}\}$.

No.	*Reactants*	*Products*	*Constant (k)*
R_1	TF + fVII	TF-fVII	$3.20 \cdot 10^6 M^{-1}s^{-1}$
R_2	TF-fVII	TF + fVII	$3.10 \cdot 10^{-2} s^{-1}$
R_3	TF + fVIIa	TF-fVIIa	$2.30 \cdot 10^7 M^{-1}s^{-1}$
R_4	TF-fVIIa	TF + fVIIa	$3.10 \cdot 10^{-5} s^{-1}$
R_5	TF-fVIIa + fVII	TF-fVIIa + fVIIa	$4.40 \cdot 10^5 M^{-1}s^{-1}$
R_6	fXa + fVII	fXa + fVIIa	$1.30 \cdot 10^7 M^{-1}s^{-1}$
R_7	fIIa + fVII	fIIa + fVIIa	$2.30 \cdot 10^4 M^{-1}s^{-1}$
R_8	TF-fVIIa + fX	TF-fVIIa-fX	$2.50 \cdot 10^7 M^{-1}s^{-1}$
R_9	TF-fVIIa-fX	TF-fVIIa + fX	$1.05 \cdot 10^{-2} s^{-1}$
R_{10}	TF-fVIIa-fX	TF-fVIIa-fXa	$6.00\ s^{-1}$
R_{11}	TF-fVIIa-fXa	TF-fVIIa + fXa	$19.00\ s^{-1}$
R_{12}	TF-fVIIa + fXa	TF-fVIIa-fXa	$2.20 \cdot 10^7 M^{-1}s^{-1}$
R_{13}	TF-fVIIa + fIX	TF-fVIIa-fIX	$1.00 \cdot 10^7 M^{-1}s^{-1}$
R_{14}	TF-fVIIa-fIX	TF-fVIIa + fIX	$2.40\ s^{-1}$
R_{15}	TF-fVIIa-fIX	TF-fVIIa + fIXa	$1.80\ s^{-1}$
R_{16}	fII + fXa	fIIa + fXa	$7.50 \cdot 10^3 M^{-1}s^{-1}$
R_{17}	fIIa + fVIII	fIIa + fVIIIa	$2.00 \cdot 10^7 M^{-1}s^{-1}$
R_{18}	fVIIIa + fIXa	fIXa-fVIIIa	$1.00 \cdot 10^7 M^{-1}s^{-1}$
R_{19}	fIXa-fVIIIa	fVIIIa + fIXa	$1.00 \cdot 10^{-4} s^{-1}$
R_{20}	fIXa-fVIIIa + fX	fIXa-fVIIIa-fX	$1.00 \cdot 10^8 M^{-1}s^{-1}$
R_{21}	fIXa-fVIIIa-fX	fIXa-fVIIIa + fX	$1.00 \cdot 10^{-5} s^{-1}$
R_{22}	fIXa-fVIIIa-fX	fIXa-fVIIIa + fXa	$8.20\ s^{-1}$
R_{23}	fVIIIa	fVIIIa$_1$-L + fVIIIa$_2$	$6.00 \cdot 10^{-5} s^{-1}$
R_{24}	fVIIIa$_1$-L + fVIIIa$_2$	fVIIIa	$2.20 \cdot 10^4 M^{-1}s^{-1}$
R_{25}	fIXa-fVIIIa-fX	fVIIIa$_1$-L + fVIIIa$_2$ + fX + fIXa	$1.00 \cdot 10^{-3} s^{-1}$
R_{26}	fIXa-fVIIIa	fVIIIa$_1$-L + fVIIIa$_2$ + fIXa	$1.00 \cdot 10^{-3} s^{-1}$
R_{27}	fIIa + fV	fIIa + fVa	$2.00 \cdot 10^7 M^{-1}s^{-1}$
R_{28}	fXa + fVa	fXa-fVa	$4.00 \cdot 10^8 M^{-1}s^{-1}$
R_{29}	fXa-fVa	fXa + fVa	$0.2\ s^{-1}$
R_{30}	fXa-fVa + fII	fXa-fVa-fII	$1.00 \cdot 10^8 M^{-1}s^{-1}$
R_{31}	fXa-fVa-fII	fXa-fVa + fII	$103.00\ s^{-1}$
R_{32}	fXa-fVa-fII	fXa-fVa + fmIIa	$63.50\ s^{-1}$
R_{33}	fXa-fVa + fmIIa	fXa-fVa + fIIa	$1.50 \cdot 10^7 M^{-1}s^{-1}$
R_{34}	fXa + TFPI	fXa-TFPI	$9.00 \cdot 10^5 M^{-1}s^{-1}$
R_{35}	fXa-TFPI	fXa + TFPI	$3.60 \cdot 10^{-4} s^{-1}$
R_{36}	TF-fVIIa-fXa + TFPI	TF-fVIIa-fXa-TFPI	$3.20 \cdot 10^8 M^{-1}s^{-1}$
R_{37}	TF-fVIIa-fXa-TFPI	TF-fVIIa-fXa + TFPI	$1.10 \cdot 10^{-2} s^{-1}$
R_{38}	TF-fVIIa + fXa-TFPI	TF-fVIIa-fXa-TFPI	$5.00 \cdot 10^7 M^{-1}s^{-1}$

R_{39}	fXa + ATIII	fXa-ATIII	$1.50{\cdot}10^3 M^{-1}s^{-1}$
R_{40}	fmIIa + ATIII	fmIIa-ATIII	$7.10{\cdot}10^3 M^{-1}s^{-1}$
R_{41}	fIXa + ATIII	fIXa-ATIII	$4.90{\cdot}10^2 M^{-1}s^{-1}$
R_{42}	fIIa + ATIII	fIIa-ATIII	$7.10{\cdot}10^3 M^{-1}s^{-1}$
R_{43}	TF-fVIIa + ATIII	TF-fVIIa-ATIII	$2.30{\cdot}10^2 M^{-1}s^{-1}$
R_{44}	fXII	fXIIa	$5.00{\cdot}10^{-4} s^{-1}$
R_{45}	fXIIa + fXII	fXIIa-fXII	$1.00{\cdot}10^8 M^{-1}s^{-1}$
R_{46}	fXIIa-fXII	fXIIa + fXII	750.00 s^{-1}
R_{47}	fXIIa-fXII	fXIIa + fXIIa	$3.30{\cdot}10^{-2} s^{-1}$
R_{48}	fXIIa + PKal	fXIIa-PKal	$1.00{\cdot}10^8 M^{-1}s^{-1}$
R_{49}	fXIIa-PKal	fXIIa + PKal	$3.60{\cdot}10^3 s^{-1}$
R_{50}	fXIIa-PKal	fXIIa + Kal	40.00 s^{-1}
R_{51}	fXII + Kal	fXII-Kal	$1.00{\cdot}10^8 M^{-1}s^{-1}$
R_{52}	fXII-Kal	fXII + Kal	45.30 s^{-1}
R_{53}	fXII-Kal	fXIIa + Kal	5.70 s^{-1}
R_{54}	PKal + Kal	Kal + Kal	$2.70{\cdot}10^4 M^{-1}s^{-1}$
R_{55}	Kal	Kal$_i$	$1.10{\cdot}10^{-2} s^{-1}$
R_{56}	fXIIa + C1inh	fXIIa-C1inh	$3.60{\cdot}10^3 M^{-1}s^{-1}$
R_{57}	fXIIa + ATIII	fXIIa-ATIII	21.60 $M^{-1}s^{-1}$
R_{58}	fXI + fIIa	fXI-fIIa	$1.00{\cdot}10^8 M^{-1}s^{-1}$
R_{59}	fXI-fIIa	fXI + fIIa	5.00 s^{-1}
R_{60}	fXI-fIIa	fXIa + fIIa	1.30 $\cdot 10^{-4} s^{-1}$
R_{61}	fXIIa + fXI	fXIIa-fXI	$1.00{\cdot}10^8 M^{-1}s^{-1}$
R_{62}	fXIIa-fXI	fXIIa + fXI	200 s^{-1}
R_{63}	fXIIa-fXI	fXIIa + fXIa	5.70 $\cdot 10^{-4} s^{-1}$
R_{64}	fXIa + fXI	fXIa + fXIa	$3.19{\cdot}10^6 M^{-1}s^{-1}$
R_{65}	fXIa + ATIII	fXIa-ATIII	$3.20{\cdot}10^2 M^{-1}s^{-1}$
R_{66}	fXIa + C1inh	fXIa-C1inh	$1.80{\cdot}10^3 M^{-1}s^{-1}$
R_{67}	fXIa + A1AT	fXIa-A1AT	$1.00{\cdot}10^2 M^{-1}s^{-1}$
R_{68}	fXIa + A2AP	fXIa-A2AP	$4.3{\cdot}10^3 M^{-1}s^{-1}$
R_{69}	fXIa + fIX	fXIa-fIX	$1.00{\cdot}10^8 M^{-1}s^{-1}$
R_{70}	fXIa-fIX	fXIa + fIX	41.00 s^{-1}
R_{71}	fXIa-fIX	fXIa + fIXa	7.70 s^{-1}
R_{72}	fIXa + fX	fIXa-fX	$1.00{\cdot}10^8 M^{-1}s^{-1}$
R_{73}	fIXa-fX	fIXa + fX	0.64 s^{-1}
R_{74}	fIXa-fX	fIXa + fXa	$7.00{\cdot}10^{-4} s^{-1}$
R_{75}	fXa + fVIII	fXa-fVIII	$1.00{\cdot}10^8 M^{-1}s^{-1}$
R_{76}	fXa-fVIII	fXa + fVIII	2.10 s^{-1}
R_{77}	fXa-fVIII	fXa + fVIIIa	0.023 s^{-1}
R_{78}	fVIIa + fIX	fVIIa-fIX	$1.00{\cdot}10^8 M^{-1}s^{-1}$

R_{79}	fVIIa-fIX	fVIIa + fIX	$0.90\ s^{-1}$
R_{80}	fVIIa-fIX	fVIIa + fIXa	$3.60{\cdot}10^{-5} s^{-1}$
R_{81}	fVIIa + fX	fVIIa-fX	$1.00{\cdot}10^{8} M^{-1} s^{-1}$
R_{82}	fVIIa-fX	fVIIa + fX	$210.00\ s^{-1}$
R_{83}	fVIIa-fX	fVIIa + fXa	$1.60{\cdot}10^{-6} s^{-1}$
R_{84}	Fbg + fIIa	Fbg-fIIa	$1.00{\cdot}10^{8} M^{-1} s^{-1}$
R_{85}	Fbg-fIIa	Fbg + fIIa	$636.00\ s^{-1}$
R_{86}	Fbg-fIIa	Fbn1 + fIIa + FPA	$84.00\ s^{-1}$
R_{87}	Fbn1 + fIIa	Fbn1-fIIa	$1.00{\cdot}10^{8} M^{-1} s^{-1}$
R_{88}	Fbn1-fIIa	Fbn1 + fIIa	$742.60\ s^{-1}$
R_{89}	Fbn1-fIIa	Fbn2 + fIIa + FPB	$7.40\ s^{-1}$
R_{90}	Fbn1 + Fbn1	$(\text{Fbn1})_2$	$1.00{\cdot}10^{6} M^{-1} s^{-1}$
R_{91}	$(\text{Fbn1})_2$	2Fbn1	$6.40{\cdot}10^{-2} s^{-1}$
R_{92}	$(\text{Fbn1})_2$ + fIIa	$(\text{Fbn1})_2$-fIIa	$1.00{\cdot}10^{8} M^{-1} s^{-1}$
R_{93}	$(\text{Fbn1})_2$-fIIa	$(\text{Fbn1})_2$ + fIIa	$701.00\ s^{-1}$
R_{94}	$(\text{Fbn1})_2$-fIIa	$(\text{Fbn2})_2$ + fIIa + FPB	$49.00\ s^{-1}$
R_{95}	Fbn2 + fIIa	Fbn2-fIIa	$1.00{\cdot}10^{8} M^{-1} s^{-1}$
R_{96}	Fbn2-fIIa	Fbn2 + fIIa	$1.00{\cdot}10^{3} s^{-1}$

Table A.6: Reaction-based model of the blood coagulation cascade (reduced version of the "Platelet-Plasma" model described in [56]). The model consists in 96 reactions among 71 molecular species. With the exception of reaction R_{44}, the values of all reaction constants were taken from [56].

Species	*Symbol*	*Concentration (M)*
α1-antitrypsin	A1AT	$4.50{\cdot}10^{-5}$
α2-antiplasmin	A2AP	$1.00{\cdot}10^{-6}$
Antithrombin III	ATIII	$3.40{\cdot}10^{-6}$
C1-inhibitor	C1inh	$2.50{\cdot}10^{-6}$
Fibrinogen	Fbg	$9.00{\cdot}10^{-6}$
Factor II	fII	$1.40{\cdot}10^{-6}$
Factor V	fV	$2.00{\cdot}10^{-8}$
Factor VII	fVII	$1.00{\cdot}10^{-8}$
Active factor VII	fVIIa	$1.00{\cdot}10^{-10}$
Factor VIII	fVIII	$7.00{\cdot}10^{-10}$
Factor IX	fIX	$9.00{\cdot}10^{-8}$
Factor X	fX	$1.6{\cdot}10^{-7}$
Factor XI	fXI	$3.10{\cdot}10^{-8}$
Factor XII	fXII	$3.40{\cdot}10^{-7}$
Prekallikrein	Pkal	$4.50{\cdot}10^{-7}$
Tissue Factor	TF	$5.00{\cdot}10^{-12}$
Tissue Factor Pathway Inhibitor	TFPI	$2.50{\cdot}10^{-9}$

Table A.7: Initial concentrations of molecular species of the blood coagulation cascade model (values taken from [56]).

A.5 Ras/cAMP/PKA pathway

In the yeast *Saccharomyces cerevisiae*, the Ras/cAMP/PKA pathway plays a major role in the regulation of metabolism, stress resistance and cell cycle progression [294, 364]. This pathway controls more than 90% of all genes that are regulated by glucose through the activation of the protein kinase A (PKA), that is able to phosphorylate a plethora of downstream proteins. PKA is activated by the binding of the second messenger cyclic-AMP (cAMP), which is synthesized by the adenylate cyclase Cyr1. The activity of Cyr1 is controlled by the monomeric GTPases Ras1 and Ras2, which cycle between a GTP-bound active state and a GDP-bound inactive state. In turn, Ras proteins are positively regulated by protein Cdc25, a Ras-GEF (Guanine Nucleotide Exchange Factor) that stimulates the GDP to GTP exchange, and negatively regulated by proteins Ira1 and Ira2, two Ras-GAP (GTPase Activating Proteins) that stimulate the GTPase activity of Ras proteins. The degradation of cAMP is governed by two phosphodiesterases, Pde1 and Pde2. These two enzymes constitute a major negative feedback in this pathway: the low-affinity phosphodiesterase Pde1 is active under the positive regulation of PKA, while the high-affinity phosphodiesterase Pde2 is active in the basal level regulation of cAMP.

The reactions describing the interactions occurring in the Ras/cAMP/PKA pathway, together with the values of the associated stochastic constants, are given in Table A.8 (see also [27, 52, 255] for further details). In particular:

- reactions $R_1, \ldots, R_{10}$ describe the switch cycle of Ras2 protein between its inactive state (Ras2-GDP) and active state (Ras2-GTP), regulated by the activity of the GEF Cdc25 and of the GAP Ira2;

- reactions R_{11}, R_{12}, R_{13} describe the synthesis of cAMP through the activation of the adenylate cyclase Cyr1, mediated by Ras2-GTP;

- reactions $R_{14}, \ldots, R_{25}$ describe the activation of PKA, mediated by the reversible binding of cAMP to its two regulatory subunits, and the subsequent dissociation of the PKA tetramer, which releases the two catalytic subunits;

- reactions $R_{26}, \ldots, R_{33}$ describe the activity of the two phosphodiesterases Pde1 and Pde2, that carry out the degradation of cAMP. The activation of Pde1 is regulated by the catalytic subunits of PKA, and it represents one of the main negative feedback control exerted by PKA within this pathway;

- reactions R_{34}, R_{35} describe the negative feedback exerted by PKA on Cdc25, whose effect is modeled as a partial inactivation of the GEF activity and a reduction of the active state level of Ras2-GTP.

The initial molecular amounts used in this thesis are summarized in Table A.9.

The SBML version of this model is available at the BioModels database [180] under submission identifier MODEL1309060000, and can be downloaded at the address http://www.ebi.ac.uk/compneur-srv/biomodels-main/BIOMD0000000478.

Table A.8: Mechanistic model of the Ras/cAMP/PKA pathway.

No.	*Reagents*	*Products*	*Stoch. constant*
R_1	Ras2-GDP + Cdc25	Ras2-GDP-Cdc25	1.0
R_2	Ras2-GDP-Cdc25	Ras2-GDP + Cdc25	1.0
R_3	Ras2-GDP-Cdc25	Ras2-Cdc25 + GDP	1.5
R_4	Ras2-Cdc25 + GDP	Ras2-GDP-Cdc25	1.0
R_5	Ras2-Cdc25 + GTP	Ras2-GTP-Cdc25	1.0
R_6	Ras2-GTP-Cdc25	Ras2-Cdc25 + GTP	1.0
R_7	Ras2-GTP-Cdc25	Ras2-GTP + Cdc25	1.0
R_8	Ras2-GTP + Cdc25	Ras2-GTP-Cdc25	1.0
R_9	Ras2-GTP + Ira2	Ras2-GTP-Ira2	$3.0{\cdot}10^{-2}$
R_{10}	Ras2-GTP-Ira2	Ras2-GDP + Ira2	$7.0{\cdot}10^{-1}$
R_{11}	Ras2-GTP + Cyr1	Ras2-GTP-Cyr1	$1.0{\cdot}10^{-3}$
R_{12}	Ras2-GTP-Cyr1 + ATP	Ras2-GTP-Cyr1 + cAMP	$2.1{\cdot}10^{-6}$
R_{13}	Ras2-GTP-Cyr1 + Ira2	Ras2-GDP + Cyr1 + Ira2	$1.0{\cdot}10^{-3}$
R_{14}	cAMP + PKA	cAMP-PKA	$1.0{\cdot}10^{-5}$
R_{15}	cAMP + cAMP-PKA	(2cAMP)-PKA	$1.0{\cdot}10^{-5}$
R_{16}	cAMP + (2cAMP)-PKA	(3cAMP)-PKA	$1.0{\cdot}10^{-5}$
R_{17}	cAMP + (3cAMP)-PKA	(4cAMP)-PKA	$1.0{\cdot}10^{-5}$
R_{18}	(4cAMP)-PKA	cAMP + (3cAMP)-PKA	$1.0{\cdot}10^{-1}$
R_{19}	(3cAMP)-PKA	cAMP + (2cAMP)-PKA	$1.0{\cdot}10^{-1}$
R_{20}	(2cAMP)-PKA	cAMP + cAMP-PKA	$1.0{\cdot}10^{-1}$
R_{21}	cAMP-PKA	cAMP + PKA	$1.0{\cdot}10^{-1}$
R_{22}	(4cAMP)-PKA	C + C + R-2cAMP + R-2cAMP	1.0
R_{23}	R-2cAMP	R + cAMP + cAMP	1.0
R_{24}	R + C	R-C	$7.5{\cdot}10^{-1}$
R_{25}	R-C + R-C	PKA	1.0
R_{26}	C + Pde1	C + Pde1p	$1.0{\cdot}10^{-6}$
R_{27}	cAMP + Pde1p	cAMP-Pde1p	$1.0{\cdot}10^{-1}$
R_{28}	cAMP-Pde1p	cAMP + Pde1p	$1.0{\cdot}10^{-1}$
R_{29}	cAMP-Pde1p	AMP + Pde1p	7.5
R_{30}	Pde1p + PPA2	Pde1 + PPA2	$1.0{\cdot}10^{-4}$
R_{31}	cAMP + Pde2	cAMP-Pde2	$1.0{\cdot}10^{-4}$
R_{32}	cAMP-Pde2	cAMP + Pde2	1.0
R_{33}	cAMP-Pde2	AMP + Pde2	1.7
R_{34}	C + Cdc25	C + Cdc25p	1.0
R_{35}	Cdc25p + PPA2	Cdc25 + PPA2	$1.0{\cdot}10^{-2}$

Table A.9: Initial molecular amounts of the Ras/cAMP/PKA model.

Molecular species	*Initial amount*
Cyr1	200
Cdc25	300
Ira2	200
Pde1	1400
PKA	2500
PPA2	4000
Pde2	6500
Ras2-GDP	20000
GDP	*$1.5 \cdot 10^6$
GTP	*$5.0 \cdot 10^6$
ATP	*$2.4 \cdot 10^7$

*The amounts of GDP, GTP and ATP are kept constant during the execution of simulations. All molecular amounts are expressed as number of molecules per cell, derived according to data presented in [97], as described in [27, 52].

Appendix B

List of abbreviations

Abbreviation	Definition
ABF	Average Best Fitness
AET	Absolute Error Tolerance in LSODA
ABM	Agent-Based Modeling
API	Application Programming Interface
BAN	Bayesian Network
BCC	Blood Coagulation Cascade
BON	Boolean Network
BDF	Backward Differentiation Formulae
CA	Cellular Automata
CC	Compute Capability
CGP	Cartesian Genetic Programming
CLE	Chemical Langevin Equation
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
CME	Chemical Master Equation
CP	Cartesian Program
CPU	Central Processing Unit
CSB	Computational Structural Biology
CT	Clotting Time
CUDA	Compute Unified Device Architecture
DE	Differential Evolution
DM	Direct Method (of SSA)

List of abbreviations

DTTS	Discrete Time Target Series
EC	Evolutionary Computation
ED	Evolutionary Design
EM	Euler's Method
ES	Evolution Strategy
EPN	Evolutionary Petri Net
FRM	First Reaction Method
GA	Genetic Algorithm
GD	Gradient Descent
GP	Genetic Programming
GPGPU	General-purpose Graphics Processing Units
GPU	Graphics Processing Units
GRM	Gene Regulation Model
GUI	Graphic User Interface
ISB	*In Silico* Biology
LSODA	Livermore Solver for Ordinary Differential Equations
MA	Memetic Algorithm
MAK	Mass-action kinetics
MD	Molecular Dynamics
MDGP	Molecular Distance Geometry Problem
MemHPG	Memetic Hybrid Particle Swarm Optimization and Genetic Algorithm
MIMD	Multiple Instruction, Multiple Data
MM	Michaelis-Menten Model
MT	Mersenne Twister
NFL	No Free Lunch
NMR	Nuclear Magnetic Resonance
NRM	Next Reaction Method
NSM	Next Subvolume Method
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
PGN	Prokaryote Gene Network
PE	Parameter Estimation
PF	Particle Filtering

PN	Petri Net
PSO	Particle Swarm Optimization
RAM	Random Access Memory
RBM	Reaction-based Model
RCGA	Real-Coded Genetic Algorithm
RE	Reverse Engineering
RET	Relative Error Tolerance in LSODA
RGSM	Randomly Generated Synthetic Model
RK4	Runge-Kutta 4
RMSD	Root Mean Square Deviation
RNG	Random Numbers Generator
RPN	Resizable Petri Net
SA	Simulated Annealing
SB	Synthetic Biology
SBML	Systems Biology Markup Language
SYSB	Systems Biology
SDE	Stochastic Differential Equation
SI	Swarm Intelligence
SIMD	Single Instruction Multiple Data
SM	Simplex Method
SMX	Streaming Multiprocessor
T&L	Transform and Lighting
TSP	Traveling Salesman Problem

Appendix C

List of symbols

This appendix provides a list of all the symbols used for the formalization throughout the whole thesis. Each symbol maintains a unique and uniform semantics in all chapters, except where it is explicitly stated otherwise.

Greek and special letters

Symbol	Definition
$\bullet$	Concatenation operator
$\circ$	Component-wise multiplication operator
α	Adaptive velocity factor in MemHPG
α_{ji}	Value in the stoichiometric matrix of reactants, corresponding to the i-th reactant of the j-th chemical reaction
β_{ji}	Value in the stoichiometric matrix of reactants, corresponding to the i-th product of the j-th chemical reaction
γ_i	i-th gene available for synthetic engineering in cuGENED
$\boldsymbol{\gamma}$	Candidate solution in cuPEPSO
$\boldsymbol{\gamma}^*$	Best solution found by cuPEPSO
$\boldsymbol{\gamma}^{\#}(IT)$	Best particle at the IT-th iteration of cuPEPSO
Γ	Set of genes available for synthetic engineering in cuGENED
δ_{ij}	Difference between measured and current distances of atoms i and j in MemHPG
Δ	Step size in deterministic simulation
Δ	Step size in GD (notation used exclusively in Section 3.1.2)

η	A generic biochemical interaction network
η_z	Best CP individual during z-th generation of CGP
ϵ	Tau-leaping error control parameter
ε	Variable associated to platelet activation status in coagSODA (notation used exclusively in Section 5.2)
ε	Error/fitness value in MemHPG
$\varepsilon^*(t)$	Smallest error value among all particles at generation t in MemHPG
ε_{max_0}	Basal activation state of platelets
ι	Number of sampling instants in cupSODA and cuTauLeaping
κ	Migration interval in cuPEPSO
κ	Number of chemical species to be sampled in cuTauLeaping (notation used exclusively in Section 5.3.1)
λ	Number of offsprings of ES
μ	EPN's mutation operator (notation used exclusively in Section 7.2)
μ	Population size in ES (notation used exclusively in Section 3.2.2)
$\mu_i(\mathbf{x})$	Quantity used by tau-leaping to determine the τ value
$\boldsymbol{\nu}_j$	State change vector of reaction R_j
ν_{ji}	Stoichiometric change of species S_i due to reaction R_j
π	Target protein for the MDGP problem
φ_i	Average error of atom i in MemHPG
$\varphi_{\mathtt{min}}$	Threshold of atom's error for the insertion into a substructure in MemHPG
$\boldsymbol{\Pi}$	Candidate solution in MemHPG
ρ	Mutation rate in CGP
ϱ	Number of runs used to calculate ABF in cuPEPSO
σ	Optimal substructure of a candidate solution in MemHPG
σ_d	d-th PSO swarm in cuPEPSO (notation used exclusively in Chapter 6)
σ_i	i-th unknown chemical species in cuGENED (notation used exclusively in Chapter 8)
$\sigma_i^2(\mathbf{x})$	Quantity used by tau-leaping to determine the τ value
Σ	Set of generic (unknown) species in cuGENED
$\boldsymbol{\theta}$	Optimal rotation in MemHPG
θ_c	Threshold for critical reactions in tau-leaping
Θ	Big theta complexity

ϑ	Velocity clamp factor in MemHPG
τ	Time step of stochastic simulation
ϕ	Putative kinetic parameterization of a candidate GRM in cuGENED
Φ	S-systems' rate constant of expression processes
$\psi_{\mathcal{S}}$	Species-to-place mapping function in PNs
ψ_{μ}	Reaction-to-place mapping function in PNs
ψ_{reac}	Reactants stoichiometry-to-PN weight mapping function in PNs
ψ_{prod}	Products stoichiometry-to-PN weight mapping function in PNs
Ψ	S-systems' rate constant of inhibitory processes
$\boldsymbol{\Upsilon}$	Optimal translation vector in MemHPG
χ	EPN's crossover operator
$\boldsymbol{\chi}$	Vector of critical reactions in cuTauLeaping
χ_a	Crossover point index in GAs
Ω	Biochemical system
Ω	Big Omega Complexity (used exclusively in the footnote on page 149)
ξ	EPN topology
Ξ	Space of possible RPN topologies

Roman letters

Symbol	**Definition**
$\mathbf{a}$	Vector of propensity functions in cuTauLeaping
$\mathbf{a}_i$	Position of i-th atom in protein $\boldsymbol{\Pi}$ in MemHPG
a_j	Propensity function of reaction R_j
a_0	Cumulative propensity
A	Number of aminoacids in a protein
$\mathbf{A}$	Flattened stoichiometric matrix of reactants in cuTauLeaping
A_{din}	Set of edges for migration in cuPEPSO
$\mathcal{A}$	Set of all vertexes connected to the current $\mathbf{x}^*$ vertex in SM
$\mathbf{b}_i$	Best position of i-th particle in PSO
B_{pg}	CUDA blocks per grid
$\mathcal{B}_{d,g}$	CUDA block in position $x = d, y = g$
$\mathbf{c}$	Vector of stochastic constants in cuTauLeaping

c_j	Stochastic constant associated to reaction R_j
C_{cog}	Social factor of PSO
C_{soc}	Cognitive factor of PSO
C	Total number of samples in cuRE
d_{ij}	Distance between atoms i and j in a protein
$d_j(\mathbf{x})$	Distinct combinations of reactants in reaction R_j according to state $\mathbf{x}$
D	Experimental conditions of target time-series in cuPEPSO
D_{ij}	Distance between atoms i and j in a candidate solution in MemHPG
$D_{\texttt{MAX}}$	Diagonal length of search space in MemHPG
$\mathcal{D}$	Number of discrete time steps in EM
E	Experimental replicates of target time-series in cuPEPSO
$\mathbf{E}$	Vector of indexes of the molecular species to be sampled in cuTauLeaping
$E(\mathbf{x})$	Energy of solution $\mathbf{x}$ in SA
F	Set of arcs in a PN
$\mathcal{F}$	Set of molecular species with constant amount in tau-leaping (notation used exclusively in Section 2.2.1)
$\mathcal{F}$	Objective function / Fitness function (according to the context)
$\mathfrak{f}$	Hill function quantifying the state of platelets activation in coagSODA
$\mathfrak{F}$	Set of functions in CGP
$\mathbf{F}$	Vector of pointers to the next time instant in cuTauLeaping
FN	Grid of functional nodes in CGP
$\mathbf{g}$	Global best of PSO
g_i	Tau-leaping additional quantity
g_{ij}	Kinetic order in S-systems
$\mathfrak{g}$	EPN's g-th generation
G	Genotype in CGP
G	Number of parallel simulations used to calculate the average dynamics in cuPEPSO (notation used exclusively in Chapter 6)
$\mathbf{G}$	Vector of auxiliary values for the calculation of τ in cuTauLeaping
$GEN_{\texttt{MAX}}$	Max number of generations in an EA
$\mathcal{G}_i$	Number of atoms for which a NMR distance from atom i is given in MemHPG
$\mathbf{h}_i$	MemHPG aggregate attractor

h_{ij}	Kinetic order in S-systems
$H(i)$	Highest order of reactions in which the chemical species S_i is involved
$\mathbf{H}$	Vector containing the highest order of reactions in cuTauLeaping
$\mathbf{H}_{\texttt{type}}$	Vector containing the type of highest order reactions in cuTauLeaping
$\mathbf{I}$	Vector of sampling time instants in cuTauLeaping
IT_{mig}	Interval between two migrations in cuPEPSO
$IT_{\texttt{MAX}}$	Maximum number of iterations in PSO and MemHPG
I_{χ}	Crossover frequency in MemHPG
k	Constant inversely proportional to the time scale of platelets activation, exploited by coagSODA
k_j	Kinetic constant associated to reaction R_j
$\mathfrak{L}$	Length of individuals in GAs
L_j	Number of firings of reaction R_j in binomial tau-leaping
$\mathcal{K}$	Dimension of a substructure σ in a protein $\mathbf{\Pi}$
K	Number of target species in PE
$\mathbf{K}$	Vector of samples of Poisson distributions in cuTauLeaping
$\mathbf{K}(\tau, \mathbf{x}, t)$	Probability distribution vector of reactions firings
$K_j(\tau, \mathbf{x}, t)$	Number of times reaction R_j will be fired in time interval $[t, t + \tau)$
$K_{\texttt{MAX}}$	Maximum capacity for places in RPNs
K_p	Maximum capacity of the p-th place in a PN
l	Levels back parameter of CGP
m	Number of places in a PN
$\mathbf{M}$	Marking of a PN
M_0	Initial marking of a PN
$\mathbf{MA}$	Stoichiometric matrix of reactants
$\mathbf{MB}$	Stoichiometric matrix of products
$\mathbf{MV}$	State change matrix
$\overline{\mathbf{MV}}$	Supplementary state change matrix
M_{p_i}	Marking of i-th place in a PN
MAX_{shared}	CUDA shared memory available on the GPU
n	Number of particles of a PSO swarm
n	Number of transitions in a PN (notation used exclusively in Sections 2.2.5 and 7.2)

List of symbols

n_c	Numer of columns in CGP
n_i	Number of input nodes in CGP
n_n	Number of input connections to each functional node in CGP
n_o	Number of output nodes in CGP
n_r	Numer of rows in CGP
n_χ	Repetitions of crossover in EPNs
N	Number of chemical species in the system
N_Γ	Number of genes available for synthetic engineering in cuGENED
N_Σ	Cardinality of the set of unknown species in cuGENED
$\mathcal{N}$	Normal distribution
$\mathfrak{N}$	Number of atoms in MDGP
O	Big O notation
$\mathcal{O}$	Set of all possible optimization problems
$\mathbf{O}$	Data structure storing all simulations in cuTauLeaping
${}^\bullet p$	Preset of a place in PNs
$p^\bullet$	Postset of a place in PNs
p_i	i-th place in a PN
p_m	Mutation probability in GAs
P	Set of places in PNs
P^h	Hidden places in RPNs
P_ξ	Set of fixed places in an EPN
$\mathcal{P}$	Population of EA
$P_j(a_j(\mathbf{x}), \tau)$	Poisson random sample with mean and variance equal to $a_j(\mathbf{x})\tau$
q_i	Halt variable in i-th thread of cuTauLeaping
Q	Population size in GAs and MemHPG
$\mathbb{Q}$	Execution flow flag in cuTauLeaping
R_c	Set of reactions marked as critical in tau-leaping
R_{nc}	Set of reactions marked as non-critical in tau-leaping
R_j	j-th reaction in a biochemical system
$\mathcal{R}$	Set of reactions in a biochemical system
S_i	i-th chemical species or component of the biological system
$\mathcal{S}$	Set of chemical species in a biochemical system
$\mathfrak{S}$	Set of optimal substructures in MemHPG

$size_{\mathtt{MAX}}$	Maximum size of substructure (in atoms) in MemHPG
$\hat{\mathcal{S}}$	Augmented set of chemical species
SH	CUDA shared memory consumption
t	Current time of the simulation
$^{\bullet}t$	Preset of a transition in PNs
$t^{\bullet}$	Postset of a transition in PNs
T	Set of transitions in PNs
T^h	Set of hidden transitions in RPNs
T	"Temperature" parameter in SA (notation used exclusively in Section 3.1.3)
t_i	i-th transition in a PN
$t_{\mathtt{MAX}}$	Maximum time of simulation in cuTauLeaping
T_d	Last time instant in condition D in cuPEPSO
T_{pb}	CUDA threads per block
T_{tot}	Total CUDA threads requested by user
$\mathcal{T}_{i,d,g}$	i-th CUDA thread in block $\mathcal{B}_{d,g}$
U	Number of threads in cupSODA and cuTauLeaping
$\mathbf{v}_i$	Velocity vector of i-th particle in PSO
$v_{\mathtt{MAX}}$	Maximum velocity of PSO particles
$\mathbf{V}$	Flattened state change stoichiometric matrix in cuTauLeaping
$\overline{\mathbf{V}}$	Flattened supplementary state change stoichiometric matrix in cuTauLeaping
w	Inertia factor for PSO
$w(p,t)$	Weight of arc from p to t in a PN
W	Weight function of PNs
$\mathbf{x}(t) \equiv \mathbf{x}$	State of the system at time t in biochemical simulation
$\mathbf{x}'$	Putative state in cuTauLeaping
$\mathbf{x}_0$	Initial state of a biochemical system
$\mathbf{x}$	Candidate solution in an optimization problem (notation used exclusively in Chapter 3)
$\mathbf{x}_i$	Position of a candidate solution in a real valued search space
$\mathbf{x}^*$	Currently visited vertex in SM
$W(t)$	Standard Wiener process depending on t

$X_k^{\gamma,d}(t_h)$	Amount of species S_k produced by a simulation using parameterization γ, calculated considering experimental condition d, at time t_h
$\langle X_k \rangle_G^{\gamma,d}(t_h)$	Average amount of species S_k calculated by performing G simulations, using parameterization γ, considering experimental condition d, at time t_h
$Y_k^{d,e}(t_h)$	Amount of species S_k in DTTS, at time t_h, during the e-th repetition in experimental condition d
Z	Number of non-zero entries in stoichiometric matrices