DOCTORAL SCHOOL
UNIVERSITY OF MILANO-BICOCCA

Department of Informatics, Systems and Communication
Ph.D. Program in Computer Science - Cycle XXXI

High-Performance Computing to Tackle Complex Problems in Life Sciences

Andrea Tangherloni

Supervisors: Prof. Daniela Besozzi
Dr. Paolo Cazzaniga

Tutor: Prof. Alberto Leporati

Ph.D. Coordinator: Prof. Stefania Bandini

Academic Year 2017-2018

To my loving family and to all people who believed in me …

Abstract

Recent advances in several research fields of Life Sciences, such as Bioinformatics, Computational Biology and Medical Imaging, are generating huge amounts of data that require effective computational tools to be analyzed, while other disciplines, like Systems Biology, typically deal with mathematical models of biochemical networks, where issues related to the lack of quantitative parameters and the efficient description of the emergent dynamics must be faced. In these contexts, High-Performance Computing (HPC) infrastructures represent a fundamental means to tackle these problems, allowing for both real-time processing of data and fast simulations. In the latest years, the use of general-purpose many-core devices, such as Many Integrated Core coprocessors and Graphics Processing Units (GPUs), gained ground. The second ones, which are pervasive, relatively cheap and extremely efficient parallel many-core coprocessors capable of achieving tera-scale performance on common workstations, have been extensively exploited in the work presented in this thesis.

Moreover, some of the problems described here require the application of Computational Intelligence (CI) methods. As a matter fact, the Parameter Estimation problem in Systems Biology, the Haplotype Assembly problem in Genome Analysis as well as the enhancement and segmentation of medical images characterized by a bimodal gray level intensity histogram can be viewed as optimization problems, which can be effectively addressed by relying on CI approaches. In the case of the Parameter Estimation problem, Evolutionary and Swarm Intelligence techniques were exploited and coupled with novel GPU-powered simulators—designed and developed in this thesis to execute both coarse-grained and fine-grained simulations—which were used to perform in a parallel fashion the biochemical simulations underlying the fitness functions required by these population-based approaches. The Haplotype Assembly and the enhancement of medical images problems were both addressed by means of Genetic Algorithms (GAs), which were shown to be very effective in solving combinatorial problems. Since the proposed approaches based on GAs are computationally demanding, a Master-Slave paradigm was exploited to distribute the workload, reducing the required running time.

The overall results show that coupling HPC and CI techniques is advantageous to address these problems and speed up the computational analyses in these research fields.

Acknowledgements

I would like to express my sincere gratitude to Professors Daniela Besozzi and Paolo Cazzaniga, they have been friends but also severe supervisors who gave me helpful suggestions and important advice during my Ph.D., they shaped me as a scientist, changing me for the rest of my life. A particular thanks to Dr. Marco S. Nobile for several reasons: he always believed in me, he helped me so many times in writing code, he spent hours and days to explain me things. Special thanks are due to Professor Giancarlo Mauri, who has been a further supportive supervisor. Many thanks go to Leonardo Rundo who has become a special friend, we grew up together during the last 3 years. Thank you for the amazing days in Nashville and Cambridge as well as for all the last minute rushes to train and bus stations since we were always late. Our adventures will continue in Cambridge during the next years. Thanks to Simone Spolaor for his friendship and for having helped me with his deep knowledge of biology and biotechnology. A big thank goes to Professor Carlos Lopez for allowing me to spend wonderful days in his research group at Vanderbilt University. I also want to send a huge "thank you" to Professor Pietro Liò who hosted me for 3 months in Cambridge, he was a special mentor who helped me during my last months with his love for sciences. Many thanks to all the wonderful colleagues who helped me during my Ph.D. visiting periods. In this regard, I am very grateful to Dr. Ana Cvejic who is investing in me for the next years at the University of Cambridge. Tons of thanks go to my new colleagues Anna, Paulina and Brynelle. You have been so gentle and have made me feel at ease from the very beginning of this new adventure. I also want to thank Professor Alberto Leporati for his support, helpful suggestions and laughter we shared. I wish to thank also the students in my lab for their friendship and for the parties we had. Finally, I am grateful to my family for its support during this three-year journey.

Table of contents

List of figures

List of tables

Introduction

Nowadays, the recent advances in several biomedical research fields, such as Bioinformatics, Computational Biology and Medical Imaging, are generating a huge amount of data on an ongoing basis [170, 461, 135]. On the other hand, different disciplines related to Life Sciences (e.g., Systems Biology) require computational methods capable of dealing with the lack of quantitative data, especially when large-scale biological systems are taken into account. Processing and analyzing this ensemble of data in a reasonable time, even in real-time, or solving the issues related to the paucity of information to investigate the functioning of complex cellular systems, are difficult tasks that can be addressed by relying on High-Performance Computing (HPC) infrastructures. In the biomedical research context, for instance, grid computing and computer clusters have been largely used due to their peculiarities that allow researchers to access global-scale resources. These infrastructures are generally flexible and characterized by a high scalability, which allows for achieving high performance by exploiting any available computational methods with minimal changes to the original code. During the latest years, the use of general-purpose many-core devices, such as Many Integrated Core (MIC) coprocessors and Graphics Processing Units (GPUs), gained ground. As a matter of fact, traditional computational methods and software tools designed and developed in research fields related to Life Sciences share a common trait: they easily become computationally demanding on Central Processing Units (CPUs), hindering their applicability in many circumstances. In order to overcome this limitation, GPUs have been widely adopted as an alternative approach to classic parallel architectures for the parallelization of computational methods in Bioinformatics, Computational Biology and Systems Biology, as extensively reviewed in [308]. GPUs were initially developed to deal with the calculations required by real-time three-dimensional computer graphics, by exploiting their underlying parallel architecture and thus freeing the CPU for further calculations [302]. Since GPUs are pervasive, relatively cheap and extremely efficient parallel many-core coprocessors, they are drawing an ever-growing attention by the scientific community. As a matter of fact, even common consumer machines are equipped with GPUs that allow the access to tera-scale performance on common workstations (and petascale performance on GPU-equipped supercomputers [220]). They can markedly decrease the running times required by traditional CPU-based software, still maintaining low-costs and energetic efficiency. However, we highlight that, in general, the implementation of computational methods able to fully exploit the peculiar architecture of GPUs is challenging, since specific programming skills are required and a complete algorithm redesign is often necessary to fully leverage the computational power of these many-core devices.

Objectives of the work

Given the effectiveness of HPC solutions, the research activity discussed in this thesis focused on the design, development and application of HPC approaches to solve computationally expensive tasks in different disciplines of Life Sciences, proposing new solutions capable of efficiently dealing with both the lack of quantitative data and the request of an effective processing of huge amounts of data. In particular, the four major problems addressed in this thesis are:

1. the definition of efficient tools to simulate and investigate the emergent dynamics of biological systems;
2. the estimation of unknown parameters of mathematical models of biochemical systems;
3. the Haplotype Assembly problem in Genome Analysis;
4. the enhancement and segmentation of medical images characterized by bimodal gray level intensity histograms.

It is worth noting that each of these problems also benefited from the use of Computational Intelligence (CI) algorithms, since they are all related to finding an optimal solution in a huge search space of candidate solutions. Evolutionary Computation (EC) [105] and Swarm Intelligence (SI) [224] are two different CI strategies that allow for measuring the quality of each candidate solution according to a specified fitness function, as in the case of mathematical optimization problems. In particular, the main concept underlying the EC methods is the evolution process that exploits genetic operators (i.e., *selection*, *mutation* and *crossover*) to evolve a population of candidate solutions. On the contrary, SI approaches rely on the emergent intelligence arising from the *collective* effort of simple agents reciprocally sharing information about the explored search space to find an optimal solution with respect to the defined fitness function. The optimization problems considered in this thesis can be partitioned into two different classes: continuous (e.g., Parameter Estimation) and discrete (e.g., Haplotype Assembly) problems, which can be effectively tackled either by means of EC or SI techniques. EC methods, like Genetic Algorithms (GAs), have been successfully applied to optimize discrete problems, while other methods, such as Particle Swarm Optimization (PSO) or Covariance Matrix Adaptation Evolution Strategy (CMA-ES), obtain better performance on real-valued problems.

All the methods based on EC and SI proposed in this thesis require a lot of fitness function evaluations, and they are employed to address problems that are intrinsically computational demanding. Since the calculations of the fitness functions are independent and some problems can be decomposed in sub-problems, parallel and distributed architectures can be used to reduce the required running time. Among all possible architectures, GPUs, MICs and multi-core CPUs have been used to accelerate the proposed methods, and the results discussed in this thesis confirm that the coupling of HPC architectures with CI represents an extremely suitable mean to achieve fast and efficient solutions to the aforementioned problems.

Figure 1 shows the main areas discussed in this thesis together with their conceptual interconnections. Namely, green hexagons represent Systems Biology and biochemical simulations; blue hexagons indicate mathematical optimization solved using EC [105] and SI [224] methods; orange hexagons symbolize the issues related to Genome Analysis; purple hexagons represent the Medical Imaging area, specifically focusing on the enhancement and segmentation of Magnetic Resonance (MR) images.

Research questions in Systems Biology

Problems 1 and 2 mentioned above pertain to Systems Biology, a multidisciplinary research field relying on the cross-talk between mathematical, computational and experimental tools. As a first step, rigorous mathematical models describing—at the desired level of detail—the complex, dynamical and non-linear nature of the biological systems must be defined. After a validation step through *ad hoc* laboratory experiments, mathematical models can be used to investigate and analyze the behavior of the biological systems in conditions that are hard or even impossible to measure with laboratory experiments. These computational analyses allow for understanding the functioning of biological systems and their response to environmental and structural perturbations [205], thus helping to formulate new hypotheses that can be tested by means of further laboratory experiments. The knowledge derived from these experiments can then be used to increase the detail of the mathematical models, as well as suggest novel research directions [229], leading to an iterative cycle of model refinements [84].

In this thesis, the conceptual framework of mechanistic modeling was taken into account to model biochemical networks. This modeling approach provides a detailed description of the molecular mechanisms that drive the interactions between the components of the biochemical network under analysis [71]. As a matter of fact, mechanistic models represent

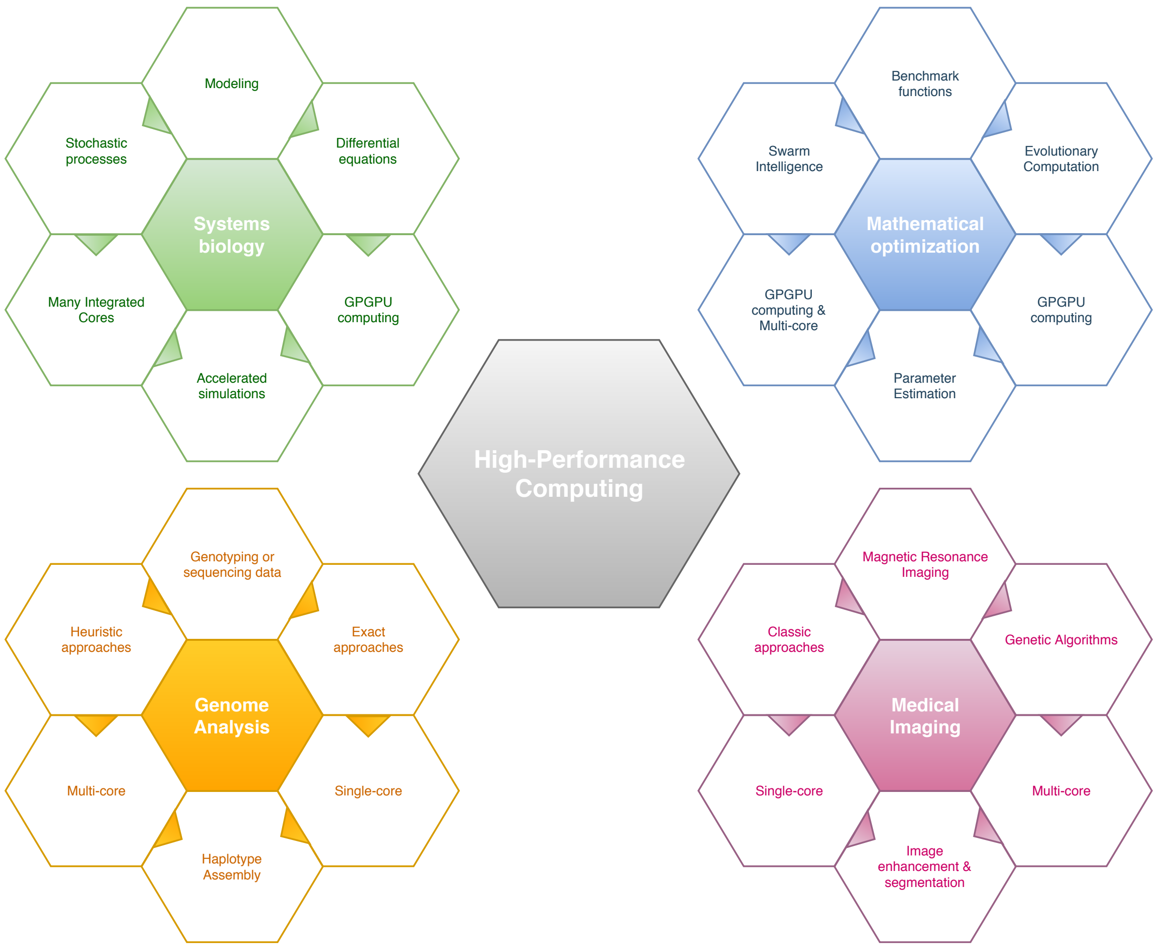


Fig. 1 Overview of the topics discussed in this thesis. Different colors have been used to represent the distint research fields and the exploited methodologies: green for Systems Biology and the tools for biochemical simulations (Chapters 1 and 4) accelerated by means of both MIC and GPUs; blue for mathematical optimization, which is the basis for the Parameter Estimation problem (Chapters 1 and 5), solved using EC and SI methods; orange for Genome Analysis (Chapters 1 and 6) that has been tackled by exploiting GAs and multi-core CPUs; purple for Medical Imaging, focusing on the enhancement and segmentation of MR images characterized by bimodal histograms (Chapters 1 and 7), addressed by means of GAs and multi-core CPUs. The EC and SI algorithms, exploited during this thesis, are presented in Chapter 2, while the HPC architecture used to accelerate the proposed methodologies are described in Chapter 3. This work aims at providing fast and reliable computational tools to achieve the aforementioned tasks, paving the way for further works integrating different disciplines for a deeper analysis of biomedical investigations.

the most prominent formalism to achieve a detailed comprehension of biological systems, since they allow for quantitative predictions of cellular dynamics. Among all possible mathematical formalizations that can be used to define mechanistic models, Reaction-Based

Models (RBMs) were considered in this thesis [41]. This choice was based on the following motivations: (*i*) RBMs are more easily readable and comprehensible with respect to other mathematical formalisms, such as differential equations. This is mostly important when experimental biologists are involved in the modeling phase and in the execution of the simulations and analyses of biological systems; (*ii*) RBMs are a flexible formalism that can be easily extended or modified during further model refinements, by simply adding or removing new reactions and/or species to the sets of reactions and chemical species previously defined; (*iii*) when an RBM is defined, it can be simulated by using both stochastic and deterministic simulators, according to the mass-action kinetics [154, 84].

The simulation of mathematical models of complex biological systems is indispensable to determine and predict the quantitative variation of the molecular species in time and in space. Simulations can be performed by relying on deterministic, stochastic or hybrid algorithms [466], which should be chosen according to the scale of the modeled system, the nature of its components and the possible role played by the biological noise. Simulations and analyses of mechanistic models can be performed if and only if a full model parameterization is properly specified, that is, all kinetic parameters and the initial conditions (i.e., molecular amounts or concentrations of the chemical species) are provided. Since a small change of even a single kinetic parameter or an initial condition can drastically modify the dynamic behavior of the system, the model parameterization should be as accurate as possible. Unfortunately, kinetic parameters are usually expensive or even impossible to measure by means of *in vivo* experiments, leading to the definition of the Parameter Estimation (PE) problem [285], which aims at the inference of accurate parameters values[1]. The most naïve, diffused, error-prone and time-consuming approach is the manually tuning of the model parameters [454]. Conversely, several automatic methods defined to identify a model parameterization that can reproduce some target dynamics can be used to obtain repeatable results. Generally, solving a PE problem consists in finding the model parameterization that allows for obtaining simulated dynamics that overlap at best some target time-series measurements of the chemical species involved in the system. This strategy is challenging since it generally leads to a non-linear, non-convex and multi-modal optimization problem [311, 432]. Moreover, an additional challenge has to be faced when data related to multiple time-series are available, each one obtained by executing different experiments under different chemico-physical conditions (e.g., temperature), replicated many times.

[1]Notice that to infer the kinetic parameters or the initial amounts of some species relying on PE procedures, it is implicit to assume that all the reactions characterizing the model are known.

Contribution In order to deal with the computational burden of biochemical simulations, we designed and developed two deterministic GPU-powered biochemical simulators. The former, named LASSIE (LArge-Scale SImulator) [425], was proposed to accelerate the simulations of large-scale biological systems, characterized by thousands reactions and molecular species. The latter, named FiCoS (Fine- and Coarse-grained Simulator) [428], was designed to reduce the running time required by the PE and other computational methods (e.g., Parameter Sweep Analysis [307] and Sensitivity Analysis [70]) that rely on a massive number of simulations of large-scale models. Moreover, a stochastic simulator based on the Stochastic Simulation Algorithm (SSA) [154] and accelerated by exploiting the MIC coprocessors [427] was proposed. These simulators allow researchers to perform simulations of biological system under physiological or perturbed conditions on common workstations.

The PE problem was tackled by exploiting several bio-inspired metaheuristics, all based on global optimization approaches. Considering the PE of small-scale models we performed a thorough analysis about the application of EC and SI approaches, showing that the PE problem is completely different to the classic benchmark functions [311, 432]. Afterwards, in order to deal with optimization problems characterized by multiple targets obtained under different experimental conditions, we proposed a multi-swarm PE approach [430] based on PSO. The PE problem of large-scale models was solved by coupling Fuzzy Self-Tuning Particle Swarm Optimization (FST-PSO) with FiCoS [442].

Research questions in Genome Analysis

Problem 3 pertains to the research field of Genome Analysis in Bioinformatics, more precisely to the reconstruction of the two distinct copies of each chromosome, called haplotypes, which is an essential step to fully characterize the genome of an individual. The computational problem of inferring the full haplotype of a cell, starting from read sequencing data, is known as Haplotype Assembly, and consists in assigning all heterozygous Single Nucleotide Polymorphisms (SNPs) to exactly one of the two chromosomes. SNPs are one of the most studied genetic variations since they play a fundamental role in many medical applications, such as drug-design, as well as in characterizing their effect on the expression of phenotypic traits [195]. Indeed, the knowledge of the complete haplotypes is generally more informative than analyzing single SNPs, especially in the study of complex disease susceptibility. Since a direct experimental reconstruction of haplotypes still requires huge sequencing efforts and is not cost-effective [240], computational approaches are extensively used to solve this problem. In particular, two classes of methods exist for the haplotype phasing [404]. The first class consists of statistical methods that try to infer the haplotypes from genotype samples in a population. These data, combined with datasets describing the frequency by which the SNPs are usually correlated in different populations, can be used to reconstruct the haplotypes of an individual. The second class of methods directly leverages sequencing data: in such a case, the main goal is to partition the entire set of reads into two sub-sets, exploiting the partial overlap among them to ultimately reconstruct the corresponding two different haplotypes of a diploid organism [333]. Among the computational methods for Haplotype Assembly, the Minimum Error Correction (MEC) is one of the most successful approaches. A weighted variant of MEC, named weighted MEC (wMEC), was proposed in [169], where the weights represent the confidence for the presence of a sequencing error and the correction process takes into account the weight associated with each SNP value of a read. These error schemes generally regard phred quality score [129], which represents the probability that a given base is called incorrectly by the sequencer, and are very valuable for processing long reads generated by third-generation sequencing technologies, as they are prone to high sequencing error rates [333]. MEC computes the two haplotypes that partition the sequencing reads into two disjoint sets with the least number of corrections to the SNP values [457], but unfortunately it was proven to be NP-hard [265]. Due to the NP-hardness of the MEC problem, some methods exploiting heuristic strategies have been proposed (see for instance [121, 239, 457, 459]).

Contribution In order to tackle the computational complexity of the haplotyping problem, we proposed GenHap [433], which can efficiently solve large instances of the wMEC problem. GenHap is a novel computational method based on GAs that yields optimal solutions by means of a global search process, without any *a priori* hypothesis about the sequencing error distribution in the reads. The computational complexity is reduced by relying on a *divide-et-impera* approach, which was distributed exploiting a Master-Slave computing paradigm that allows for speeding up the required computations, reducing the computational burden.

Research questions in Medical Imaging

Problem 4 pertains to the research field of Medical Imaging, which plays a key role in the clinical workflow thanks to its capability of representing anatomical and physiological features that are otherwise inaccessible for inspection, thus proposing accurate imaging biomarkers and clinically useful information [367, 243]. Due to the appearance of the depicted objects as well as the information conveyed by the pixels, medical images are considerably different from the pictures usually analyzed in Pattern Recognition and Computer Vision. As a matter of fact, medical imaging techniques exploit several different principles to measure spatial distributions of physical attributes of the human body, allowing us to better understand complex or rare diseases [441]. The effectiveness of these techniques can be reduced by a lot of phenomena, such as noise and partial volume effect [441], which might affect the measurement processes involved in imaging and data acquisition devices. Image contrast and details might also be impaired by the procedures used in medical imaging, as well as by the physiological nature of the body part under investigation. Moreover, medical images actually convey an amount of information related to high image resolution and high pixel depth, which could overwhelm the human vision capabilities in distinguishing among dozens of gray levels [325]. Thus, improvements in the appearance and visual quality of medical images are essential to allow physicians to attain valuable information that would not be immediately observable in the original image, and assisting them in anomaly detection, diagnosis, and treatment. In this context, image enhancement techniques aim at realizing a specific improvement in the quality of a medical image: the enhanced image is expected to better reveal certain features, compared to their original appearance [104].

In the clinical routine, Contrast-Enhanced (CE) MRI is a diagnostic technique that enables a more precise assessment of the imaged tissues, resulting the most prominent modality to obtain soft-tissue imaging [54], especially in oncology, since it provides significant improvements—in terms of image contrast and resolution—between lesion and healthy tissue [288]. However, MRI data are affected by acquisition noise [417] and are also prone to imaging artifacts, related to magnetic susceptibility and large intensity inhomogeneities of the static magnetic field (i.e., streaking or shadowing artifacts [33]), especially by using high magnetic field strengths. These aspects make MR image enhancement a challenging task devoted to improve the outcome of automatic segmentation. Medical image segmentation concerns both detection and delineation of anatomical or physiological structures from the background, distinguishing among the different components included in the image [27]. This allows for the extraction of clinically useful information and features in medical image analysis [367, 218]. Accordingly, computer-assisted approaches enable quantitative imaging [122], aiming at accurate and objective measurements from digital images regarding a Region of Interest (ROI) [482, 243]. Indeed, image segmentation is still one of the most challenging research areas especially in medical image analysis [122]. Accurately delineating the ROIs is a critical task, since manual segmentation procedures are time-expensive, error-prone, and operator-dependent (i.e., not ensuring result repeatability).

Contribution The existing image enhancement approaches generally attempt to improve the contrast level of the whole image and do not address the issues related to overlapped gray level intensities; by so doing, neither the region contour sharpness nor the image thresholding results can be improved. However, determining the best pre-processing of an image—able to preserve the structural information of the image while enhancing the underlying bimodal distribution of the histogram bins—is a complex task on a multi-modal fitness landscape. For this reason, we proposed MedGA [379], a novel image enhancement technique based on GAs that aims at strengthening the sub-distributions in medical images with an underlying bimodal histogram of the gray level intensities. We developed also a Master-Slave version of MedGA to distribute on multiple cores the analysis of the batch of slices obtained by the CE-MRI of a single patient.

Thesis structure

This thesis is structured as follows. In the first part (Chapters 1–3) the prerequisites necessary for the development of the proposed solutions are introduced. In the second part (Chapters 4–7), the novel computational approaches proposed in the thesis, together with their applications, are described in details. These chapters constitute the original contribution of the research activities proposed in this thesis. The last chapter finally provides a discussion of this work and future research directions. More precisely:

- In Chapter 1, the complex problems addressed in this thesis are introduced. In the first two sections, the principal modeling approaches for biochemical networks, along with the simulations methodologies, are described. In this context, the role of model parameters is shown and the PE problem defined. Afterwards, the Haplotype Assembly problem is discussed and, finally, an introduction to medical images characterized by a bimodal histogram is provided. For each problem, the main state-of-the-art approaches are also presented.

- Chapter 2 starts with a description of classic and local optimization techniques. Then, the most known EC and SI techniques are presented. Among them, GAs, PSO, and two improved version of PSO relying on Fuzzy Logic are discussed in more details, since the former was used to tackle the Haplotype Assembly and the enhancement of MR images, while the latter to address the PE problem.

- In Chapter 3, the main features of the most known HPC paradigms are described, analyzed and discussed, starting from traditional architectures (grid computing and compluter clusters) to many-core solutions (MICs and GPUs). General-Purpose GPU computing is described in details along with the Compute Unified Device Architecture (CUDA), which has been exploited to develop both LASSIE and FiCoS.

- In Chapter 4, we describe the two deterministic GPU-powered biochemical simulators (i.e., LASSIE and FiCoS), as well as the stochastic simulator accelerated by means of MICs coprocessors. We show, in particular, that FiCoS drastically reduces the running time required by the Parameter Sweep Analysis of two real RBMs. We also propose an empirical analysis that might facilitate the selection of proper HPC architectures to parallelize SSA, depending on the number of required independent simulations.
- Chapter 5 describes the PE problem and its solution by means of EC and SI techniques. A comparison of the efficiency of different EC and SI techniques in solving benchmark functions with respect to the PE problem is also presented [311, 432]. Then, a PE methodology designed to deal with the availability of experimental data measured in multiple initial conditions is discussed. Finally, a PE methodology combining FiCoS and a Fuzzy Logic-based version of PSO is presented to solve the PE of large-scale models of biological systems [442].
- In Chapter 6, GenHap is described in details along with the obtained results. The results show that GenHap always obtains high accuracy solutions (in terms of haplotype error rate), and is faster than a state-of-the-art approach, considering both short and long reads generated by second and third-generation sequencing technologies, respectively. We also assessed the performance of GenHap on two different real datasets, showing that future-generation sequencing technologies can highly benefit from GenHap, thanks to its capability of efficiently solving large instances of the Haplotype Assembly problem.
- Chapter 7 focuses on MedGA. First, the method for image enhancement and its results are presented, showing that MedGA is capable of outperforming the state-of-the-art approaches. Afterwards, a segmentation pipeline based on MedGA is proposed and the achieved results discussed in details. The results highlight that applying MedGA as a pre-processing step, the MR image segmentation accuracy is considerably increased, allowing for measurement repeatability in clinical workflows.
- In the last chapter some conclusive remarks about the presented works are given. Possible improvements and future directions are also discussed.
- In Appendix A the RBMs of some real biochemical systems, exploited to test the methodologies proposed in this thesis, are described.

Scientific production

The work presented in this thesis is based on the following publications[2]:

Journal papers

- Tangherloni A., Spolaor S., Rundo L., Nobile M.S., Cazzaniga P., Mauri G., Liò P., Merelli I., Besozzi D.: **GenHap: A Novel Computational Method Based on Genetic Algorithms for Haplotype Assembly**, BMC Bioinformatics, Special Issue of the 17th Workshop on Network Tools and Applications for Biology (NETTAB 2017), BioMed Central, in press

- Rundo L., Tangherloni A., Nobile M.S., Militello C., Besozzi D., Mauri G., Cazzaniga P.: **MedGA: A novel evolutionary method for image enhancement in medical imaging systems**, Expert Systems With Applications, 119, pp. 387-399, Elsevier, 2018

- Tangherloni A., Nobile M.S., Besozzi D., Mauri G., Cazzaniga P.: **LASSIE: simulating large-scale models of biochemical systems on GPUs**, BMC Bioinformatics, 18(1), BioMed Central, 2017

- Tangherloni A., Nobile M.S., Cazzaniga P., Besozzi D., Mauri G.: **Gillespie's Stochastic Simulation Algorithm on MIC coprocessor**, The Journal of Supercomputing, 73(2), pp. 1-11, Springer, 2017

- Nobile M.S., Cazzaniga P., Tangherloni A., Besozzi D.: **Graphics Processing Units in Bioinformatics, Computational Biology and Systems Biology**, Briefings in Bioinformatics, 18(5), pp. 1-16, Oxford University Press, 2017

Book chapters

- Tangherloni A., Rundo L., Spolaor S., Nobile M.S., Merelli I., Besozzi D., Mauri G., Cazzaniga P., Liò P.: **High-Performance Computing for haplotyping: Models and platforms**, In *Euro-Par 2018: Parallel Processing Workshops*, Lecture Notes in Computer Science, 11339, pp. 650-661, Springer

[2]A number of additional topics related to biomedical images were also subject of my research during the Ph.D. program [374, 380, 370, 378, 371, 373]. These works, which mainly present applications of the methods presented here, have not been included in this thesis for space limits.

- Beccuti M., Cazzaniga P., Pennisi M., Besozzi D., Nobile M.S., Pernice S., Russo G., Tangherloni A., Pappalardo F.: **GPU accelerated analysis of Treg-Teff cross regulation in relapsing-remitting multiple sclerosis**, In *Euro-Par 2018: Parallel Processing Workshops*, Lecture Notes in Computer Science, 11339, pp. 626-637, Springer

- Cazzaniga P., Nobile M.S., Tangherloni A., Besozzi D.: **Accelerating stochastic simulations of mechanistic models of biological systems: Advantages and issues in the parallelization on Graphics Processing Units**, In *Quantitative Biology: Theory, Computational Methods, and Models*, MIT Press, 2018

Conference proceedings

- Spolaor S., Tangherloni A., Rundo L., Nobile M.S., Cazzaniga P.: **Estimation of Kinetic Reaction Constants: Exploiting Reboot Strategies to Improve PSO's Performance**, Proceedings of the 14th International Meeting on Computational Intelligence Methods for Bioinformatics and Biostatistics (CIBB 2017), Cagliari (Italy), 2019

- Nobile M.S., Tangherloni A., Rundo L., Spolaor S., Besozzi D., Mauri G., Cazzaniga P.: **Computational Intelligence for Parameter Estimation of Biochemical Systems**, proceedings of IEEE Congress on Evolutionary Computation (CEC 2018), Rio de Janeiro (Brazil), pp. 1-8, 2018

- Tangherloni A., Rundo, L., Spolaor, S., Cazzaniga, P., Nobile, M. S.: **GPU-Powered Multi-Swarm Parameter Estimation of Biological Systems: A Master-Slave Approach**, proceedings of the 26th Euromicro IEEE International Conference on Parallel, Distributed and Network-based Processing (PDP 2018), Cambridge (UK). Special Session on: Parallel and distributed high-performance computing solutions in Systems Biology, pp. 698-705, 2018

- Spolaor S., Tangherloni A., Rundo L., Nobile M.S., Cazzaniga P.: **Reboot Strategies in Particle Swarm Optimization and their Impact on Parameter Estimation of Biochemical Systems**, proceedings of IEEE International Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB 2017). Special Session on: Parallel and Distributed High Performance Computing Solutions for Computational Intelligence Methods, Manchester (UK), pp. 1-8, 2017

- Tangherloni A., Rundo L., Nobile M.S.: **Proactive Particles in Swarm Optimization: a Settings-Free Algorithm for Real-Parameter Single Objective Optimization Problems**, proceedings of IEEE Congress on Evolutionary Computation (CEC 2017). Special Session & Competitions on: Real-Parameter Single Objective Optimization, Donostia - San Sebastian (Spain), pp. 1940-1947, 2017

- Tangherloni A., Nobile M.S., Cazzaniga P.: **GPU-powered Bat Algorithm for the Parameter Estimation of biochemical kinetic values**, proceedings of IEEE International Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB 2016). Special Session on: Parallel and distributed high performance computing solutions for computational intelligence methods, Chiang Mai (Thailand), pp. 1-6, 2016

- Nobile M.S., Tangherloni A., Besozzi D., Cazzaniga P.:**GPU-powered and Settings-Free Parameter Estimation of Biochemical Systems**, proceedings of IEEE Congress on Evolutionary Computation (CEC 2016). Special Session on: Computational Intelligence Methods Accelerated on Parallel and Distributed Architectures for Applications in Bioinformatics, Computational Biology and Systems Biology, Vancouver (Canada), pp. 32-39, 2016

Submitted papers

- Tangherloni A., Spolaor S., Cazzaniga P., Besozzi D., Rundo L., Mauri G., Nobile M.S.: **Benchmark functions do not capture the complexity of biochemical Parameter Estimation**, submitted to Applied Soft Computing, Elsevier, 2018

- Rundo L., Tangherloni A., Cazzaniga P., Nobile M.S., Russo G., Gilardi M.C., Vitabile S., Mauri G., Besozzi D., Militello C.: **Genetic Algorithms improve thresholding-based segmentation of bimodal Magnetic Resonance Images**, submitted to Computer Methods and Programs in Biomedicine, Elsevier, 2018

- Riva S.G., Tangherloni A., Nobile M.S., Cazzaniga P., Besozzi D.: **SMGen: A novel Generator of synthetic models of Biological Systems**, under preparation for BMC Bioinformatics, BioMed Central

- Tangherloni A., Nobile M.S., Cazzaniga P., Capitoli G., Spolaor S., Rundo L., Mauri G., Besozzi D.: **FiCoS: a fine- and coarse-grained GPU-powered deterministic simulator for biochemical networks**, under preparation for Scientific Reports, Nature

Chapter 1

Complex problems in Life Sciences

In this chapter, the main concepts and state-of-the-art methods about the four problems addressed during this thesis (i.e., the simulation of (large-scale) biochemical models, the Parameter Estimation (PE) problem of biochemical systems, the Haplotype Asssembly, the enhancement and segmentation of medical images characterized by bimodal histograms) are introduced and explained in details. In the first three sections, we present (*i*) the different modeling approaches that can be exploited to describe biological systems, (*ii*) the simulation methodologies that can be used to investigate the dynamic behaviors of the modeled systems, and (*iii*) the fundamental role played by the parameters on the emergent behaviors of biological systems, along with the PE problem. In the following section, the Haplotye Assembly problem, which is one of the most hot topics in Bioinformatics and Genome Analysis, is introduced. Finally, in the last section we provide a description of the main issues related to the enhancement and the segmentation of medical images characterized by bimodal histograms.

1.1 Modeling biochemical systems

The representation of a biological system by means of a mathematical formulation, which allows for understanding the phenomenon of interest, must take into account the scale of the modeled system, the nature of its components, and the possible role played by the biological noise. The chosen mathematical formulation should be capable of integrating the different kinds of data obtained from laboratory experiments, as well as providing new hypotheses to be tested *in vivo*. As a matter of fact, the mathematical formulation should allows us to: (*i*) investigate the functioning of complex biological systems; (*ii*) characterize their emergent properties exploiting the interactions among their fundamental components [10]; (*iii*) predict how these complex biological systems might behave in both physiological and perturbed conditions.

Biochemical reaction networks can be described by relying on different modeling approaches, depending on the desired level of details. A plethora of computational methods can then be used to simulate and analyze the properties of the modelled system. The choice of the most appropriate mathematical and computational approaches is related to the purpose of the study, and should take into account the specific biological problem under investigation, as well as the complementary analysis performed by means of laboratory experiments. In order to define the most appropriate mathematical model, the following factors must be carefully considered: (1) the purpose (i.e., the scientific question) underlying the model; (2) the expected information that should be collected from *in silico* analyses; (3) the available data and their quality. These factors allow for properly determining the abstraction level of the model that we should define and use.

The modeling approaches can be partitioned into two classes: (*i*) coarse-grained models (e.g., interaction-based or constraint-based models), and (*ii*) fine-grained (e.g., mechanism-based) models. The former class is suitable for the identification of the main components or modules of the system under investigation, but they generally do not take into account the majority of quantitative and kinetic properties of the system, resulting too often poor of biochemical details. The latter class is composed of mechanism-based modeling approaches that are characterized by the highest predictive capabilities about the functioning of the system at molecular level. Nevertheless, they require detailed kinetic information about the interactions between the molecular species occurring in the system. The lack of these data may limit the applicability of such detailed modeling approaches. Figure 1.1 depicts a complete overview of interaction-based, constraint-based and mechanism-based modeling approaches, along with their main characteristics.

The interaction-based modeling approaches exploit graphs to describe biochemical networks: the nodes represent the chemical species, while the edges represent the existing connections between two species by means of some kind of biochemical or functional interactions. Due to their simplicity and the low computational burden, this methodology is suited for modeling large-scale networks (e.g., genome-wide), a task that has become more and more suitable also thanks to the last high-throughput technologies, which generate a huge amount of data about living systems [407]. The main analyses that can be performed on these interaction graphs are related to the topological characteristics of the network, such as shortest paths, degree distribution or clustering coefficient, allowing for determining the main features of the structural organization of the network at the large-scale level. These analyses are suitable for understanding the processes underlying the evolution of the network

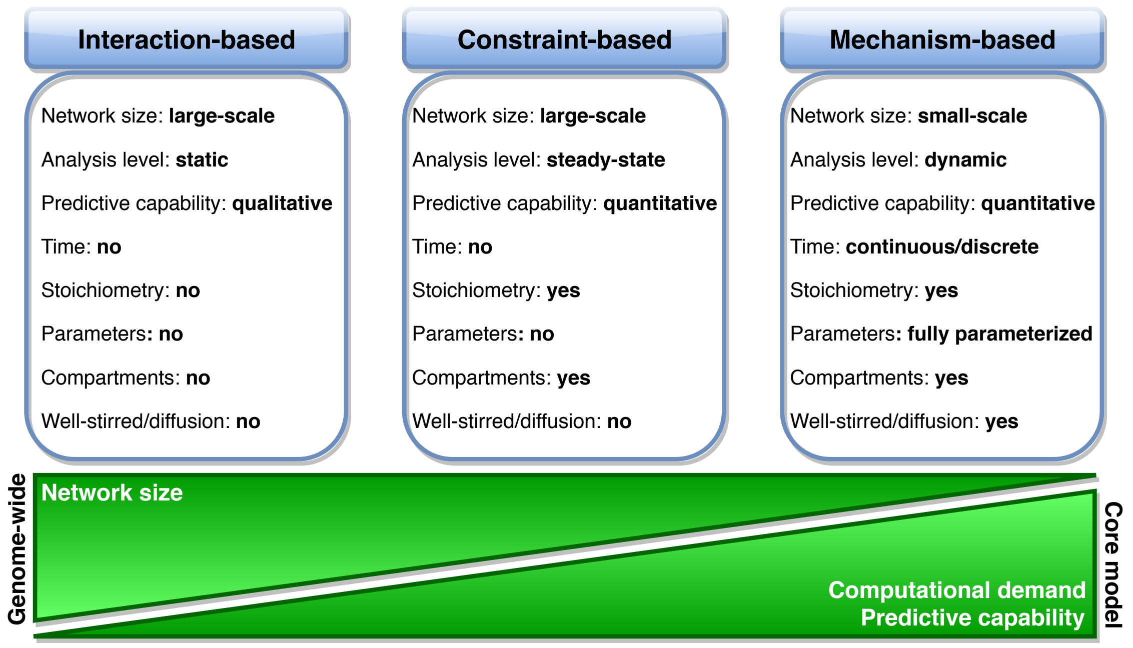

Interaction-based
Network size: large-scale
Analysis level: static
Predictive capability: qualitative
Time: no
Stoichiometry: no
Parameters: no
Compartments: no
Well-stirred/diffusion: no
Constraint-based
Network size: large-scale
Analysis level: steady-state
Predictive capability: quantitative
Time: no
Stoichiometry: yes
Parameters: no
Compartments: yes
Well-stirred/diffusion: no
Mechanism-based
Network size: small-scale
Analysis level: dynamic
Predictive capability: quantitative
Time: continuous/discrete
Stoichiometry: yes
Parameters: fully parameterized
Compartments: yes
Well-stirred/diffusion: yes
Genome-wide
Network size
Computational demand
Predictive capability
Core model

Fig. 1.1 Figure adapted from [71] showing the principal modeling approaches as well as their main characteristics and differences. These modeling approaches can be partitioned in coarse-grained (interaction-based, constraint-based) and fine-grained (mechanism-based) models, which vary in terms of: (*i*) size of the system (i.e., number of components and respective interactions); (*ii*) computational demand required for *in silico* analysis; (*iii*) predictive capabilities. The interaction-baseds approaches allow for defining genome-wide models, while the mechanism-based approaches are more suitable for core models. Regarding the analysis, interaction-based models are generally exploited for the static and qualitative investigation of the topological properties of the network; constraint-based models are used to study the quantitative flux distributions at steady-state; due to their fully parameterization, mechanism-based models are the best choice for the quantitative analysis of the system dynamics.

structure [262], providing new insights about the components at the basis of the robustness and redundancy in the biochemical network [29]. Bayesian Networks (BANs) [335] represent another type of interaction-based models. In such a case, the vertices of the directed acyclic graph are random variables, while the arcs represent conditional dependencies between the vertices. BANs are a powerful framework to design probabilistic relationships between the components (e.g., genes and their mutual regulations) of the biochemical system under investigation. However, since in BANs it is not possible to include loops and feedback mechanisms [56], they remain rarely used.

Constraint-based models include some quantitative information in addition to the network structure previously discussed. Constraint-based models are generally exploited to explore the set of possible flux distributions (i.e., flows of metabolites [324]) in large-scale metabolic networks. Among all possible sources of information, reaction stoichiometry (explained in details in Section 1.1.1) represents the basic one that can be considered. However, this information does not allow for fully determining the feasible states of the system. In order to limit the search space of the candidate solutions of flux distributions, further constraints should be exploited. The most used constraints are given by transcriptomic and enzyme capacity, as well as thermodynamic constraints associated to the reversibility of the reactions. In literature, several techniques have been proposed to the aim of investigating the feasible flux distributions at steady-state. It is worth mentioning Flux Balance Analysis [324], Extreme Pathway Analysis [348, 386], and Elementary Mode Analysis [444]. Since the constraints are generally modeled as linear equations, the Simplex Method (described in Section 2.1.1) is probably the most used approach to solve the resulting Linear Programming problem that aims at identifying the optimal state of the system.

Mechanism-based models are generally used to quantitatively describe the system at the level of functional biochemical interactions, obtaining the highest predictive capability regarding the cellular dynamics. Due to this peculiarity, they represent the best solution to achieve a detailed comprehension of biological systems [71], as they allow for reproducing the temporal evolution of all the molecular species occurring in the model. Despite their high predictive capability, they can be become easily unfeasible to be used due to the required computational burden of the simulations and analyses that increases along with the number of the components and interactions composing the system under investigation. In addition, mechanism-based modeling approaches require the complete knowledge of quantitative parameters (i.e., reaction stoichiometry, initial molecular concentrations/number of molecules of the chemical species, and kinetic constants). Unfortunately, these parameters are difficult or even impossible to measure by means of *in vivo* and *ad hoc* experiments. Thus, the lack of these data can limit the applicability of this modeling approach. Mechanistic models can be defined by means of different mathematical formalism, as described in Section 1.1.1.

1.1.1 Mechanistic modeling

The biochemical reaction networks taken into account in this thesis are assumed to obey the mass-action kinetic (MAK) law [84, 78, 455, 301], which is the fundamental and empirical law governing biochemical reaction rates. MAK states that in a diluted solution in dynamic equilibrium, the rate of an elementary reaction is directly proportional to the product of the concentrations of its reactants, raised to the power of the corresponding stoichiometric coefficients [78, 301]. Since MAK is the most general framework to describe the biochemical kinetics, it is at the basis of the modeling approaches, simulation tools and PE methods designed and developed in this thesis. In addition, all the biological systems considered in what follows are assumed to be well-stirred, at thermal equilibrium and characterized by a fixed volume.

Reaction-based models

Given a biochemical system Ω, the corresponding Reaction-Based Model (RBM) is defined by specifying the set of N molecular species $\{S_1,\ldots,S_N\}$ and the set of M biochemical reactions $\{R_1,\ldots,R_M\}$. A generic reaction is described as follows:

$$R_i : \sum_{j=1}^{N} a_{ij}S_j \xrightarrow{k_i} \sum_{j=1}^{N} b_{ij}S_j, \quad i = 1,\ldots,M, \tag{1.1}$$

where a_{ij}, $b_{ij} \in \mathbb{N}$ are the stoichiometric coefficients of the reactants and products species—that is, the number of molecules of species S_j that are consumed or produced when the reaction takes place, respectively—and $k_i \in \mathbb{R}^+$ is the kinetic constant associated with R_i. The set of reactions $\{R_1,\ldots,R_M\}$ can be written compactly in the matrix-vector form:

$$\mathbf{AS} \xrightarrow{\mathbf{K}} \mathbf{BS},$$

where $\mathbf{S} = [S_1 \cdots S_N]^T$ is the N-dimensional column vector of molecular species, $\mathbf{K} = [k_1 \cdots k_M]^T$ is the M-dimensional column vector of kinetic constants, and $\mathbf{A},\mathbf{B} \in \mathbf{N}^{M\times N}$ are the so-called stoichiometric matrices whose (non-negative) elements $[A]_{i,j}$ and $[B]_{i,j}$ correspond to the stoichiometric coefficients a_{ij} and b_{ij} of the reactants and the products of all reactions, respectively. Besides $\mathbf{A},\mathbf{B}$ we can defined the state change matrix $\mathbf{H} = \mathbf{A} - \mathbf{B}$ associated to the system, whose rows $\mathbf{h}_i \in \mathbb{Z}$, with $\mathbf{h}_i = (h_{i1},\ldots,h_{iN}) = (b_{i1} - a_{i1},\ldots,b_{iN} - a_{iN})$, are the state change vectors. Notice that each vector $\mathbf{h}_i$ represents the stoichiometric change of the species S_j due to the reaction R_i. We also denote by $\overline{\mathbf{H}}$ the state change matrix composed of the elements $\overline{h}_{i,j} = 0$ of the species $S_j \in \mathbf{F}$, where $\mathbf{F} \subset \mathbf{S}$ is the subset of species whose amounts is kept fixed during the simulation of the dynamics of the whole system. This strategy allows for simulating the non-limiting availability of some chemical resources, and can be used to mimic the execution of *in vitro* experiments where some species are continually introduced in Ω to keep their amount constant [82].

Null reactions (i.e., $b_{ij} = a_{ij} = 0$ for all $j = 1,\ldots,N$, indicated as $\emptyset \xrightarrow{k_i} \emptyset$), are not taken into account in this thesis, as well as reactions in the form R_i: $a_jS_j \xrightarrow{k_i} b_jS_j$, for any a_j and b_j. As a matter of fact, reactions in that form correspond to unfeasible biochemical processes that convert a_j molecules of the species S_j in b_j molecules of the same species. On the contrary, source and degradation reactions are considered feasible. A source reaction R_i (denoted as $\emptyset \xrightarrow{k_i} products$) is characterized by $a_{ij} = 0$, for all $j = 1, \ldots, N$, while a degradation reaction (denoted as $reagents \xrightarrow{k_i} \emptyset$) is obtained by setting $b_{ij} = 0$, for all $j = 1, \ldots, N$.

The state of the system Ω at time t is defined as $\mathbf{x}(t) = (x_1(t), \ldots, x_N(t))$, where x_j represents the amount of the species S_j at time t. In RBMs, the amount of chemicals composing the biochemical system can be given either as concentrations (i.e., $x_j \in \mathbb{R}$) or number of molecules (i.e., $x_j \in \mathbb{N}$). Concentrations are used to perform deterministic simulations, while number of molecules are suitable for stochastic simulations. In the latter case, the value k_i is usually indicated by c_i and represents the stochastic constant associated to the reaction R_i (a real value representing the physical and chemical properties of R_i [153]). The fundamental hypothesis underlying the stochastic formulation of chemical kinetics states that the average probability of the reaction R_i to occur in the interval $(t,\ t + dt)$ is equal to $c_i dt$ [153]. The dynamics of the system can then be calculated by using a stochastic simulation algorithm, as described in Section 1.2.2, taking into account the probabilities of all the reactions.

Notice that since a reaction simultaneously involving more than two reactants has a probability to take place almost equal to zero, in this thesis only first and second-order reactions (i.e., at most two reactant molecules of the same or different species can appear in the left hand side of Equation 1.1) are considered. For this reason, the matrices **A** and **B** are sparse.

Differential equations

Ordinary Differential Equations (ODEs) represents the traditional mechanistic modeling approach in Systems Biology. The dynamics (i.e., rate of changes) of the chemical species can be investigated by describing a biochemical reaction network Ω by means of a system of coupled ODEs. For instance, starting from the following biochemical reactions involving four species, namely S_1, S_2, S_3 and S_4:

$$S_1 + S_2 \underset{k_2}{\overset{k_1}{\rightleftarrows}} S_3 \overset{k_3}{\rightarrow} S_4, \tag{1.2}$$

the following system of coupled ODEs can be derived by assuming the MAK law:

$$\begin{cases} \frac{dx_1}{dt} = -k_1x_1x_2 + k_2x_3 \\ \frac{dx_2}{dt} = -k_1x_1x_2 + k_2x_3 \\ \frac{dx_3}{dt} = k_1x_1x_2 - k_2x_3 - k_3x_3 \\ \frac{dx_4}{dt} = k_3x_3 \end{cases}, \quad (1.3)$$

where x_j represents the concentration value of species S_j[1].

More generally, given an arbitrary RBM it is always possible to obtain the corresponding system of ODEs as follows:

$$\frac{d\mathbf{x}}{dt} = (\mathbf{B} - \mathbf{A})^T[\mathbf{k} \odot \mathbf{x}^{\mathbf{A}}], \quad (1.4)$$

where $\mathbf{A}$ and $\mathbf{B}$ are the stoichiometric coefficient matrix of the reactants and products, respectively, $\mathbf{k}$ is the column vector of the kinetic constants, $\mathbf{x}$ is the column vector of the concentration values, and $\mathbf{x}^{\mathbf{A}}$ denotes the vector-matrix exponentiation form [78], where the symbol $\odot$ denotes the entry-by-entry matrix multiplication (Hadamard product). Formally, $\mathbf{x}^{\mathbf{A}}$ is a M-dimensional vector whose i-th component is given by $x_1^{A_{i1}} \cdots x_N^{A_{iN}}$, for $i = 1, \dots, M$. It is worth noting that each ODE appearing in Equation 1.4 is a polynomial function consisting in at least one monomial, which is associated with a specific kinetic constant.

ODE models can be easily generalized by exploiting the Reaction Rate Equations (RREs) [18]. Assuming that there exist N different chemical species involved in M different reactions, for each species S_j the following ODE is derived:

$$\frac{dx_j}{dt} = \sum_{i=1}^{M} h_{ij}\alpha_i(\mathbf{x}, k_i), \text{ for } j = 1, \dots N, \quad (1.5)$$

where α_i is the so-called propensity function of the reaction R_i, defined according to the reactant concentrations (contained in the state vector $\mathbf{x}$) and the kinetic parameters of the reaction. For instance, given the biochemical system defined in Equation 1.1.1, according to the MAK law, since the reaction R_1 involves the species S_1 and S_2, its propensity is $-k_1x_1x_2$.

ODE models represent a powerful mathematical framework to simulate biochemical systems. However, the dynamics obtained by means of ODE solvers represent an approximation of the system. As a matter of fact, ODE models do not take into account the stochasticity that usually characterizes several biochemical systems [444]. As a way of example, many

[1]The ODE models described in this section could obviously exploit other biochemical kinetic laws, such as Michaelis-Menten kinetic [301] or Hill functions, which are not explicitly taken into account in this thesis

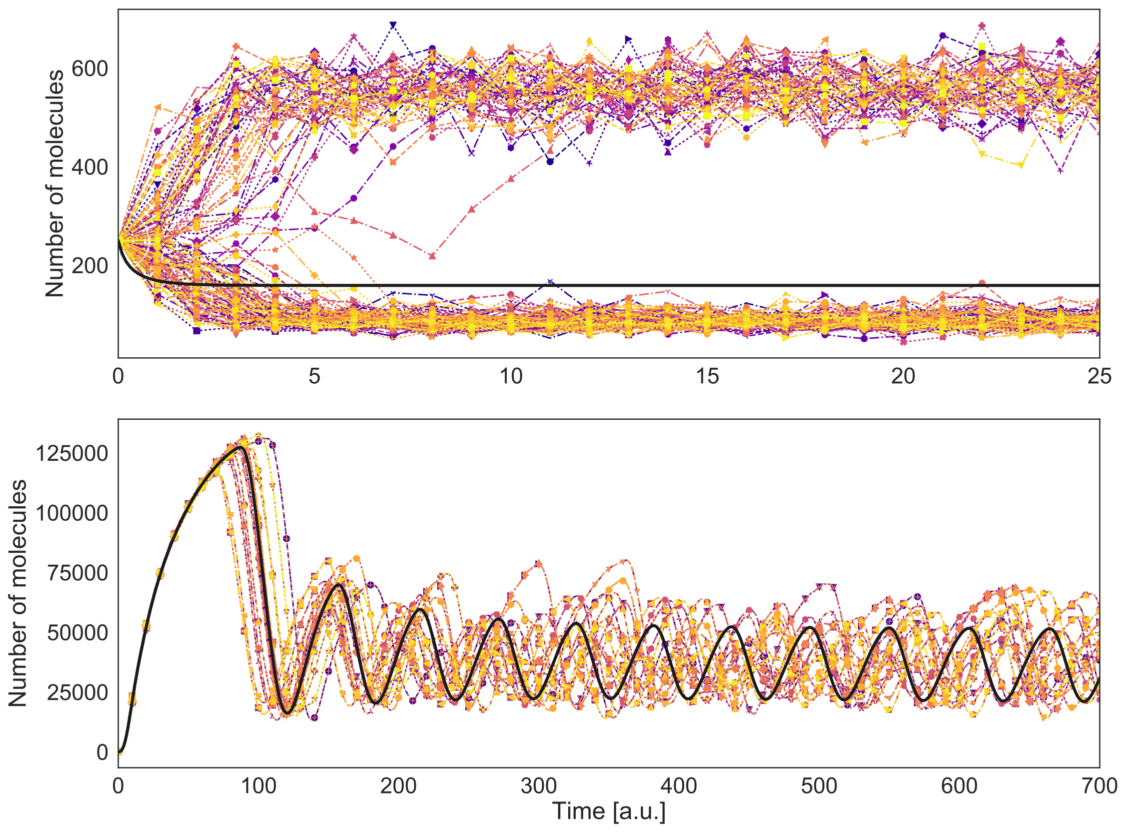


Fig. 1.2 Example of bistable (top panel, Schlögl model) and oscillatory (bottom panel, Ras/cAMP/PKA model) dynamics. In both cases, a deterministic simulation (black line) and 128 stochastic simulations (colored dotted lines) are compared. The Schlögl model (see Appendix A.6) is a simple biochemical system characterized by bistability, in which two steady states are reached starting from the same initial condition. The deterministic simulation can reach only one of the possible states, while stochastic simulations allow for reproducing the bistability characterizing this model. The Ras/cAMP/PKA model (see Appendix A.5) plays a major role in the regulation of metabolism in the yeast *Saccharomyces cerevisiae*. This model is characterized by oscillatory dynamics that can be correctly reproduced by using both deterministic and stochastic simulations. As a matter of fact, the deterministic simulation is able to reproduce the average trend of the dynamics.

cellular regulation networks are characterized by molecular species occurring in very low amounts in the cell and, as such, they are often affected by noise [127]. Moreover, non deterministic behaviors can arise from the randomness at the molecular scale, which produces stochastic phenomena at the macromolecular scale (e.g., bistability). Since classical ODE approaches are not capable of capturing the effects of stochastic processes, they cannot be straightforwardly exploited to simulate and analyze bistability phenomena (see Figure 1.2, top panel), while stochastic approaches [173] are more suitable for an effective investigation of macromolecular phenomena of this type. On the contrary, ODE models can be effectively and efficiently used to simulate and analyze other kind of emergent dynamics, such as oscillatory systems (see Figure 1.2, bottom panel), thanks to their lower computational load with respect to that required by stochastic approaches.

An extension of ODEs is represented by Stochastic Differential Equations (SDEs), such as the Chemical Langevin Equations (CLEs) [156, 158]. These modeling approaches allow for reproducing the dynamics of the systems by adding noise terms to the rate equations; still, CLEs can only yield an approximation of the correct dynamics of the system under investigation. CLEs can be properly used where the concentration of chemical species corresponds to a high number of molecules. In such a case, the system can be modeled by means of the following stochastic equation:

$$\mathbf{x}(t+\tau) = \mathbf{x}(t) + \sum_{i=1}^{M} \mathbf{h}_i \mathrm{N}(\alpha_i(\mathbf{x}(t))\tau, \alpha_i(\mathbf{x}(t))\tau), \tag{1.6}$$

where $\mathrm{N}(\alpha_i(\mathbf{x}(t))\tau, \alpha_i(\mathbf{x}(t))\tau)$ is a normally distributed random variable with mean $\mu = \alpha_i(\mathbf{x}(t))\tau$ and variance $\sigma^2 = \alpha_i(\mathbf{x}(t))\tau$. Nevertheless, when either the amount of molecules is low (e.g., lower than 100) or the number of reactions that occurs in the time interval $(t, t+\tau)$ is small, CLEs do not represent a good modeling solution.

Chemical Master Equation

The Chemical Master Equation (CME) can be exploited to model a biochemical system Ω when a stochastic approach is required to investigate Ω, by describing the probability distribution function associated to Ω itself [449]. The probability that the system will be in a state $\mathbf{x}$ at time t, starting from an initial state $\mathbf{x}_0$ at time t_0, is indicated as $P(\mathbf{x}, t|\mathbf{x}_0, t_0)$. It can be calculated by applying the CME as follows:

$$\frac{\partial P(\mathbf{x}, t|\mathbf{x}_0, t_0)}{\partial t} = \sum_{i=1}^{M} \Big(\alpha_i(\mathbf{x} - \mathbf{v}_i) P(\mathbf{x} - \mathbf{v}_i, t|\mathbf{x}_0, t_0) - \alpha_i(\mathbf{x}) P(\mathbf{x}, t|\mathbf{x}_0, t_0) \Big), \tag{1.7}$$

where $\mathbf{h}_i$ is the state change associated to the reaction R_i and $\alpha_i(\mathbf{x})$ is the propensity function of R_i. Given an arbitrary couple of initial state $\mathbf{x}_0$ and initial time t_0, CME allows for exactly calculating the probability that the system will be in the state $\mathbf{x}$ at time t. However, the analytical solution of the CME is generally untractable. In Section 1.2.2 it will be shown how to deal with this issue.

Rule-based models

Rule-based models represent an extension of RBMs suitable for dealing with the combinatorial complexity that derives from multiple protein-protein interactions, which generally cause an explosion of the reactions as well as of the intermediate molecular complexes [196]. This modeling approach has a higher level of abstraction with respect to RBMs: the molecules are represented by means of “objects" with extended features, while the chemical reactions are defined as rules involving these objects, instead of specific chemical species. This strategy allows for reducing both the size and complexity of the model under investigation [197].

In order to simulate a rule-based model, two main approaches exist. The former consists in re-expanding the model by explicitly enumerating all the chemical species and reactions [280], obtaining an RBM that can be simulated by applying a stochastic or a deterministic method. This approach generates large-scale models characterized by hundreds or thousands of species and reactions, which are generally difficult to simulate. The latter relies on “network-free" Monte Carlo [480] methods, by explicitly representing all the molecules composing the system. However, this strategy cannot be applied in case of large-scale models due to the required prohibitive spatial computational complexity. Some “hybrid" particle-counters approaches have been proposed to deal with this issue: they represent the chemical species that appear in large numbers by means of normal variables [87].

In this thesis, the rule-based modeling is used to generate large-scale RBMs that are then simulated by means of FiCoS (Fine- and Coarse-grained Simulator) [428], as described in Section 4.3.

1.2 Simulation algorithms for mechanism-based models

Given a biochemical system modeled by exploiting one of the mechanistic approaches described in the previous section, its dynamics can be obtained by using a simulation methodology, which is generally strongly related to the chosen modeling formalism. Notice that it is not sufficient to describe the biochemical system as a simple interaction network to perform the simulation of a mechanistic model. As a matter of fact, both deterministic and stochastic simulation techniques require that a proper kinetic parameterization and the initial state of the system are known, which must be provided as part of the model. In the following sections the most common simulation techniques are described in details.

1.2.1 Deterministic simulation

Deterministic simulations rely on ODE modelling approach: given the system of ODEs, along with a set of kinetic constants as well as an initial state at time t_0, the temporal evolution of the biological system in the interval $[t_0, t_{max}]$ can be obtained by means of a numerical integration algorithms.

We remind that, given a system of N ODEs (one for each species occurring in the biochemical system), its Cauchy problem is defined as:

$$\begin{cases} \frac{d\mathbf{x}}{dt} &= f(t, \mathbf{x}) \\ \mathbf{x}(t_0) &= \mathbf{x}_0 \end{cases}, \tag{1.8}$$

where $\mathbf{x}(t) = (x_1(t), x_2(t), \ldots, x_{\mathrm{N}}(t))$ is the state of the system, $x_j(t)$ represents the concentration of species S_j at time t, and $t \in [t_0, t_{max}]$.

Euler method

The most straightforward and basic numerical algorithm for solving systems of ODEs is the Euler's method (EM), defined by the mathematician Euler in 1768 [59]. It is an iterative algorithm in which the differential equation is considered as a simple formula to calculate the slope of the tangent to the unknown curve described by the system of ODEs. To be more precise, in the case of a single ODE $x'(t) = f(t, x(t))$, given the initial point $x(t_0) = x_0$ and a fixed step-size h, the ODE is used to calculate the slope of the curve in each point, by exploiting the value of the previous point as follows:

$$x_{n+1} = x_n + hf(t_n, x_n),$$

where $x_n \approx x(t_n)$. Note that x_n is an approximation of the real solution to the ODE at time t_n. EM can be derived by considering the Taylor expansion of the function f around a point t_0:

$$x(t_0 + h) = x(t_0) + hx'(t_0) + \frac{1}{2}h^2 x''(t_0) + O(h^3).$$

EM is obtained by substituting x' with $f(t, x)$, which corresponds to the differential equation definition, and ignoring the quadratic and higher-order terms. EM results in a first-order, explicit, single- and fixed-step approximation method, whose error at each time t is $O(h^2)$.

Besides the explicit version of the EM, we can derive the implicit version [67] (also called Backward EM) of this simple method as follows:

$$x_{n+1} = x_n + hf(t_{n+1}, x_{n+1}).$$

Since the new approximation x_{n+1} appears on both sides of the equation, a non-linear equation for the unknown term x_{n+1} must be solved during each iteration of the method.

Backward EM is the simplest Backward Differentiation Formulae (BDF), since it represents the BDF of order 1. The general formula for a BDF can be written as

$$\sum_{i=0}^{q} \alpha_i \mathbf{x}(t - t_i) = h\beta_0 f(t, \mathbf{x}(t)), \tag{1.9}$$

where the coefficients α_i (with $\alpha_0 = 1$) and β_0 are chosen according to the order q of BDF [438], and h is user-defined. Note that, for $q > 6$, the absolute stability region of the resulting BDF methods is too small, so that these BDFs are characterized by numerical instability [147]. Therefore, BDFs with an order q greater than 6 are not used. Considering a system of ODEs, since each BDF is an implicit method, at each time step it requires the solution of a non-linear system of equations, which can be solved by using the iterative Newton–Raphson method [35]. This system can be written as follows:

$$g(\mathbf{x}(t)) \equiv \mathbf{x}(t) - h\beta_0 f(t, \mathbf{x}(t)) + c^{\mathbf{x}}(t) = 0, \tag{1.10}$$

where $c^{\mathbf{x}}(t) = \sum_{i=1}^{q} \alpha_i \mathbf{x}(t - t_i)$ is a constant quantity depending on previous values of the state of the system $\mathbf{x}(t)$ and on the order q.

The Newton–Raphson method allows for finding successively better approximations z of the zeros of a real-valued function $f(z) = 0$, and it is repeated until a sufficiently accurate value is reached. The approximation at iteration n is calculated as follows:

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}. \tag{1.11}$$

This idea can be extended to a system of non-linear equations, by using the Jacobian matrix $\mathbf{J}(t, \mathbf{x}(t))$ of $f(t, \mathbf{x}(t))$. Therefore, the following system must be solved:

$$\begin{cases} \mathbf{J}(\mathbf{x}^i)\mathbf{v}\mathbf{x}^i = -f(\mathbf{x}^i) \\ \mathbf{x}^{i+1} = \mathbf{x}^i + \mathbf{v}\mathbf{x}^i \end{cases}, \tag{1.12}$$

where $\mathbf{vx}^i$ is the vector used to update $\mathbf{x}^i$, and $\mathbf{x}^{i+1}$ at time t can be written as

$$\mathbf{x}^{i+1}(t) = \mathbf{x}^i(t) - \left(\mathbf{I} - \frac{\partial f}{\partial \mathbf{x}}\right)^{-1} g(\mathbf{x}^i(t)). \tag{1.13}$$

Since the calculations required to invert the matrix $\left(\mathbf{I} - \frac{\partial f}{\partial \mathbf{x}}\right)$ are computationally expensive, we can derive the following linear system:

$$\begin{cases} (\boldsymbol{I} - h\beta_0 \frac{\partial \boldsymbol{f}}{\partial \mathbf{x}}(t, \mathbf{x}^i(t))\boldsymbol{\Delta v}^i = -g(t, \mathbf{x}^i(t)) \\ \mathbf{x}^{i+1}(t) = \mathbf{x}^i(t) + \boldsymbol{\Delta v}^i \end{cases}, \tag{1.14}$$

where $\boldsymbol{\Delta v}^i$ is the vector solution of the linear system (Equation 1.14) at iteration i, and $\mathbf{I}$ is the identity matrix. The $\boldsymbol{\Delta v}^i$ vector is used to update the iteration vector $\mathbf{x}^{i+1}(t)$ required by the Newton-Raphson method.

Since the evaluation of the Jacobian matrix at each iteration is computationally expensive, the modified Newton–Raphson method [403] can be exploited to reduce the computational load. Thus, the iteration matrix is evaluated once at the beginning of each step, based on the predicted value $\mathbf{x}^0(t)$, and it is used for all the iterations during the current step. The linear system can be solved by using a linear system solver, such as the LU factorization method [30]. The Newton-Raphson method is iterated until the maximum number of iterations is reached, or a sufficiently accurate value is achieved (i.e., smaller than a user-defined tolerance value ε_{NR}). When this method ends, the state of the system is updated as $\mathbf{x}(t+h) = \mathbf{x}^{i+1}(t)$.

Runge-Kutta methods

The mathematicians Runge and Kutta introduced a family of methods to mitigate the error of EM, by extending the basic idea of the EM itself [59]. A generic Runge-Kutta (RK) method of order r—the value describing the desired local truncation error, i.e., $O(h^{r+1})$—and s stages is defined as:

$$\mathbf{x}(t_{n+1}) \simeq \mathbf{x}_{n+1} = \mathbf{x}_n + h\sum_{i=1}^{s} \beta_i l_i, \tag{1.15}$$

The auxiliary variables l_i are given by the following relationship:

$$l_i = f(t_n + c_i h, \mathbf{x}_n + h\sum_{j=1}^{s-1} \alpha_{ij} l_j), \text{with } i = 1, \ldots, s. \tag{1.16}$$

Table 1.1 Butcher tableau of a generic Runge-Kutta method.

$$\begin{array}{c|cccc} c_1 & \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1s} \\ c_2 & \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & \alpha_{s1} & \alpha_{s2} & \cdots & \alpha_{ss} \\ \hline & \beta_1 & \beta_2 & \cdots & \beta_s \\ & \beta_1^* & \beta_2^* & \cdots & \beta_s^* \end{array} = \begin{array}{c|c} \mathbf{c} & \mathbf{\Lambda} \\ \hline & \boldsymbol{\beta}^T \end{array}$$

The coefficients α_{ij}, β_i and c_i allow for characterizing every RK method, which can be represented in the so-called Butcher tableau (see Table 1.1).

The RK methods are partitioned into three classes based on the coefficients represented in their Butcher tableau. To be more precise, the following rules are applied:

- if the matrix $\mathbf{\Lambda}$ is lower triangular (i.e., $\alpha_{ij} = 0$ for $j > i$, with $i, j = 1, \dots, s$), then the method is said to be *explicit*;
- if the matrix $\mathbf{\Lambda}$ is lower triangular including the main diagonal, the method is called *semi-implicit*;
- conversely, the method is *implicit*. In such a case, in order to calculate the auxiliary variables l_i, with $i = 1, \dots, s$, at least one non-linear system must be solved. As a matter of fact, each l_i depends on all l_j, with $j = 1, \dots, s$.

Generally, implicit RK methods are exploited for the resolution of stiff problems due to their stability regions. Stiffness is a well-known phenomen characterizing the numerical solution of ODEs. It is a subtle, difficult and important concept that depends on the ODE to be solved, the initial conditions, as well as the numerical method taken into account. An ODE is considered stiff if the sought solution varies slowly, but at the same time nearby solutions varying rapidly exist. In this condition, explicit methods may use small integration step-size to obtain satisfactory results, increasing the required running time. Systems of ODEs related to biochemical systems are often affected by stiffness [194]. Note that a system of biochemical reactions may be stiff when two well-separated dynamical modes, determined by fast and slow reactions, occur [158].

The most simple RK method is the midpoint method (also called RK2), which is a second-order method with two stages. Considering a single ODE, its Butcher tableau is

Table 1.2 Butcher tableau of the midpoint method, which corresponds to a two stages Runge-Kutta method of second-order.

$$\begin{array}{c|cc} 0 & & \\ \frac{1}{2} & \frac{1}{2} & \\ \hline & 0 & 1 \end{array}$$

shown in Table 1.2, which corresponds to:

$$x_{n+1} = x_n + hf\Big(t_n + \frac{1}{2}h, x_n + \frac{1}{2}hf(t_n, x_n)\Big).$$

In such a case, the slope is calculated twice: (*i*) the slope of the curve is calculated from t_n to $t_n + h$; (*ii*) this value is used to obtain the slope at the midpoint $t_n + \frac{1}{2}h$. The midpoint method has a local error at each step of order $O(h^3)$, while the global error is of order $O(h^2)$.

The most popular explicit single- and fixed-step RK method is called RK4, which is a fourth-order method. According to its Butcher tableau (see Table 1.3), the following quantities must be calculated:

$$\begin{aligned}
l_1 &= f(t_n, x_n), \\
l_2 &= f\Big(t_n + \frac{1}{2}h, x_n + \frac{1}{2}hl_1\Big), \\
l_3 &= f\Big(t_n + \frac{1}{2}h, x_n + \frac{1}{2}hl_2\Big), \\
l_4 &= f\Big(t_n + h, x_n + hl_3\Big).
\end{aligned}$$

Starting from these quantities, the new point x_{n+1} is calculated as the weighted average of l_1, l_2, l_3 and l_4:

$$x_{n+1} = x_n + \frac{1}{6}h(l_1 + 2l_2 + 2l_3 + l_4).$$

RK4 has a final error of order $O(h^4)$, which allows for obtaining an improved approximation of the unknown curve.

In contrast to the multi-step methods, RK methods are single-step methods in which it is possible to change the step-size during the ODE resolution, according to the desired error, which is controlled by means of two tolerances (i.e., absolute and relative tolerances). Adaptive RK methods are characterized by two embedded methods, as shown in the Butcher tableau (Table 1.1) with β_i and β_i^*, with $i = 1, \ldots, s$.

Table 1.3 Butcher tableau of the RK4 method, which is a fourth-order Runge-Kutta method.

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

The most known explicit adaptive RK methods are: (*i*) the Runge–Kutta–Fehlberg (RKF) [134, 277, 176, 133] method that implements two methods of orders 5 and 4, respectively; (*ii*) the Dormand–Prince (DOPRI) method [116, 115, 176]. To be more precise, in this thesis we exploited the DOPRI5 version, which is a method of order 5. Both RKF and DOPRI5 are capable of varying the integration step-size during the resolution of the ODE system by exploiting their embedded methods.

Runge–Kutta–Fehlberg. Given a single ODE to be solved, according to the Butcher tableau given in Table 1.4, RKF requires two different approximated solutions u_{n+1} and w_{n+1}, which are generated as follows:

$$\begin{aligned} u_{n+1} &= x_n + \frac{25}{216}l_1 + \frac{1408}{2565}l_3 + \frac{2197}{4104}l_4 - \frac{1}{5}l_5, \\ w_{n+1} &= x_n + \frac{16}{135}l_1 + \frac{6656}{12825}l_3 + \frac{28561}{56430}l_4 - \frac{9}{50}l_5 + \frac{2}{55}l_6, \end{aligned} \tag{1.17}$$

where

$$\begin{aligned} l_1 &= hf(t,x_n), \\ l_2 &= hf\Big(t_n + \frac{1}{4}h, x_n + \frac{1}{4}l_1\Big), \\ l_3 &= hf\Big(t_n + \frac{3}{8}h, x_n + \frac{3}{32}l_1 + \frac{9}{32}l_2\Big), \\ l_4 &= hf\Big(t_n + \frac{12}{13}h, x_n + \frac{1932}{2197}l_1 - \frac{7200}{2197}l_2 + \frac{7296}{2197}l_3\Big), \\ l_5 &= hf\Big(t_n + h, x_n + \frac{439}{216}l_1 - 8l_2 + \frac{3680}{513}l_3 - \frac{845}{4104}l_4\Big), \\ l_6 &= hf\Big(t_n + \frac{1}{2}h, x_n - \frac{8}{27}l_1 + 2l_2 - \frac{3544}{2565}l_3 + \frac{1859}{4104}l_4 - \frac{11}{40}l_5\Big). \end{aligned} \tag{1.18}$$

Table 1.4 Butcher tableau of the RKF method.

0						
$\frac{1}{4}$	$\frac{1}{4}$					
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$				
$\frac{12}{13}$	$\frac{1932}{2197}$	$\frac{-7200}{2197}$	$\frac{7296}{2197}$			
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$\frac{-845}{4104}$		
$\frac{1}{2}$	$\frac{-8}{27}$	2	$\frac{-3544}{2565}$	$\frac{1859}{4104}$	$\frac{-11}{40}$	
	$\frac{25}{216}$	0	$\frac{1408}{2565}$	$\frac{2197}{4104}$	$\frac{-1}{5}$	0
	$\frac{16}{135}$	0	$\frac{6656}{12825}$	$\frac{28561}{56430}$	$\frac{-9}{50}$	$\frac{2}{55}$

Table 1.5 Butcher tableau of the DOPRI5 method.

0							
$\frac{1}{5}$	$\frac{1}{5}$						
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$					
$\frac{4}{5}$	$\frac{44}{45}$	$\frac{-56}{15}$	$\frac{32}{9}$				
$\frac{8}{9}$	$\frac{19372}{6561}$	$\frac{-25360}{2187}$	$\frac{64448}{6561}$	$\frac{-212}{729}$			
1	$\frac{9017}{3168}$	$\frac{-355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$\frac{-5103}{18656}$		
1	$\frac{35}{834}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$\frac{-2187}{6784}$	$\frac{11}{84}$	
	$\frac{35}{834}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$\frac{-2187}{6784}$	$\frac{11}{84}$	0
	$\frac{5179}{57600}$	0	$\frac{7571}{16695}$	$\frac{393}{640}$	$\frac{-92097}{339200}$	$\frac{187}{2100}$	$\frac{1}{40}$

To evaluate the accuracy of u_{n+1} and w_{n+1}, a user-defined tolerance $\varepsilon \in \mathbb{R}$ is exploited. Two additional values, $ER \in \mathbb{R}$ and $\delta \in \mathbb{R}$, are calculated as follows:

$$ER = \frac{|w_{n+1} - u_{n+1}|}{h}, \quad \delta = 0.84\left(\frac{\varepsilon}{ER}\right)^{\frac{1}{4}}. \tag{1.19}$$

If $ER \leq \varepsilon$, then u_{n+1} is accepted as new solution, that is, $x_{n+1} = u_{n+1}$; otherwise, the solutions u_{n+1} and w_{n+1} are rejected and recalculated by using a new step-size, which is computed as $h = h \cdot \delta$. Thus, RKF is a method with an error estimator of order $O(h^5)$.

Dormand-Price. Similar to RKF, given a single ODE to be solved, DOPRI5 requires two different approximated solutions u_{n+1} and w_{n+1} (see the Butcher tableau given in Table 1.5), which are generated as follows:

$$
\begin{aligned}
u_{n+1} &= x_n + \frac{35}{834}l_1 + \frac{500}{1113}l_3 + \frac{125}{192}l_4 - \frac{2187}{6784}l_5 + \frac{11}{84}l_6, \\
w_{n+1} &= x_n + \frac{5179}{57600}l_1 + \frac{7571}{16695}l_3 + \frac{393}{640}l_4 - \frac{92097}{339200}l_5 + \frac{187}{2100}l_6 + \frac{1}{40}l_7,
\end{aligned}
\tag{1.20}
$$

where

$$
\begin{aligned}
l_1 &= hf(t, x_n), \\
l_2 &= hf\Big(t_n + \frac{1}{5}h, x_n + \frac{1}{5}l_1\Big), \\
l_3 &= hf\Big(t_n + \frac{3}{10}h, x_n + \frac{3}{40}l_1 + \frac{9}{40}l_2\Big), \\
l_4 &= hf\Big(t_n + \frac{4}{5}h, x_n + \frac{44}{45}l_1 - \frac{56}{15}l_2 + \frac{32}{9}l_3\Big), \\
l_5 &= hf\Big(t_n + \frac{8}{9}h, x_n + \frac{19372}{6561}l_1 - \frac{25360}{2187}l_2 + \frac{64448}{6561}l_3 - \frac{212}{729}l_4\Big), \\
l_6 &= hf\Big(t_n + h, x_n + \frac{9017}{3168}l_1 - \frac{355}{33}l_2 + \frac{46732}{5247}l_3 + \frac{49}{176}l_4 - \frac{5103}{18656}l_5\Big), \\
l_7 &= hf\Big(t_n + h, x_n + \frac{35}{834}l_1 + \frac{500}{1113}l_3 + \frac{125}{192}l_4 - \frac{2187}{6784}l_5 + \frac{11}{84}l_6\Big).
\end{aligned}
\tag{1.21}
$$

Once the solutions of the two embedded methods are calculated, they are used to evaluated the committed error. If the error is greater than the desirable error—which strongly depends on the user-defined absolute and relative tolerances—the current solutions are rejected and re-calculated using a smaller step-size; otherwise, u_{n+1} is accepted as new solution, that is, $x_{n+1} = u_{n+1}$. In both cases, the solutions of the two embedded methods are also used to calculate the step-size for the next iteration. The absolute tolerance is a threshold below which the value of the solution is considered unimportant. The absolute error tolerance determines the accuracy when the solution approaches zero. The relative tolerance is a measure of the error relative to the size of the solution. Approximately, this value controls the number of correct digits in the solution, except those smaller than the absolute tolerance. DOPRI5 is a method with an error estimator of order $O(h^6)$.

Radau IIA family. Among the implicit adaptive RK methods, the Radau IIA family [177, 178] is probably the most known. The most exploited Radau IIA method is the so-called RADAU5 (Table 1.6 shows its Butcher tableau), which is a method of order 5.

Since RADAU5 is implicit, given a single ODE to be solved it requires the resolution of one non-linear equation for each l_i (with $i = 1, \dots, s$) that must be calculated. Since $\beta_1^* = 0$ and $\beta_2^* = 0$, four non-linear equations will be solved during each iteration of the method. In

Table 1.6 Butcher tableau of the RADAU5 method.

$\frac{4-\sqrt{6}}{10}$	$\frac{88-7\sqrt{6}}{360}$	$\frac{296-169\sqrt{6}}{1800}$	$\frac{-2+3\sqrt{6}}{225}$
$\frac{4+\sqrt{6}}{10}$	$\frac{296+169\sqrt{6}}{1800}$	$\frac{88+7\sqrt{6}}{360}$	$\frac{-2-3\sqrt{6}}{225}$
1	$\frac{16-\sqrt{6}}{36}$	$\frac{16+\sqrt{6}}{36}$	$\frac{1}{9}$
	$\frac{16-\sqrt{6}}{36}$	$\frac{16+\sqrt{6}}{36}$	$\frac{1}{9}$
	0	0	1

the case of a system of ODEs, four non-linear systems must be solved during each iteration of the method. Notice that the Jacobian matrix of the system of ODEs is required to solve the non-linear systems. The first three non-linear systems are converted into three linear systems exploiting the modified Newton–Raphson method [403] and then solved by means of the LU factorization method [30]. Notice that in each iteration of the Newton–Raphson method, these three linear systems must be solved. The last non-linear system is converted into a linear system using a linear combination of the l_i (with $i = 1,\dots,s$) obtained at the end of Newton–Raphson method. This linear system is then solved exploiting the LU factorization method.

Once the solutions of the two embedded methods are calculated, they are used to evaluated the committed error. If the error is greater than the desirable error, which depends on the user-defined absolute and relative tolerances, the current solutions are rejected and re-calculated by using a smaller step-size; otherwise, they are accepted. In both cases, the solutions of the two embedded methods are also used to calculate the step-size for the next iteration.

Notice that in both DOPRI5 and RADAU5 the maximum number of internal iterations allowed for the integration can be modified by the user.

Figure 1.3 depicts the comparison of the aforementioned methods to solve the differential equation $x'(t) = -0.5e^{\frac{t}{2}}\sin(5t) + 5e^{\frac{t}{2}}\cos(5t) + x(t)$ with initial condition $x(0) = 0$, whose exact solution is $x(t) = e^{\frac{t}{2}}sin(5t)$. This simple example shows that fixed step-size methods (explicit EM, implicit EM, RK2 and RK4) are capable of obtaining a good approximation of the exact solution only when a correct step-size is selected (top panel). As a matter of fact, using a step-size too large (bottom panel) the error is propagated during the iterations. Even though implicit EM resulted less accurate with respect to RK4 in this simple example, it is suitable for a stiff ODE (or a system of ODEs), while RK4 might require a very small step-size to obtain accurate solutions.

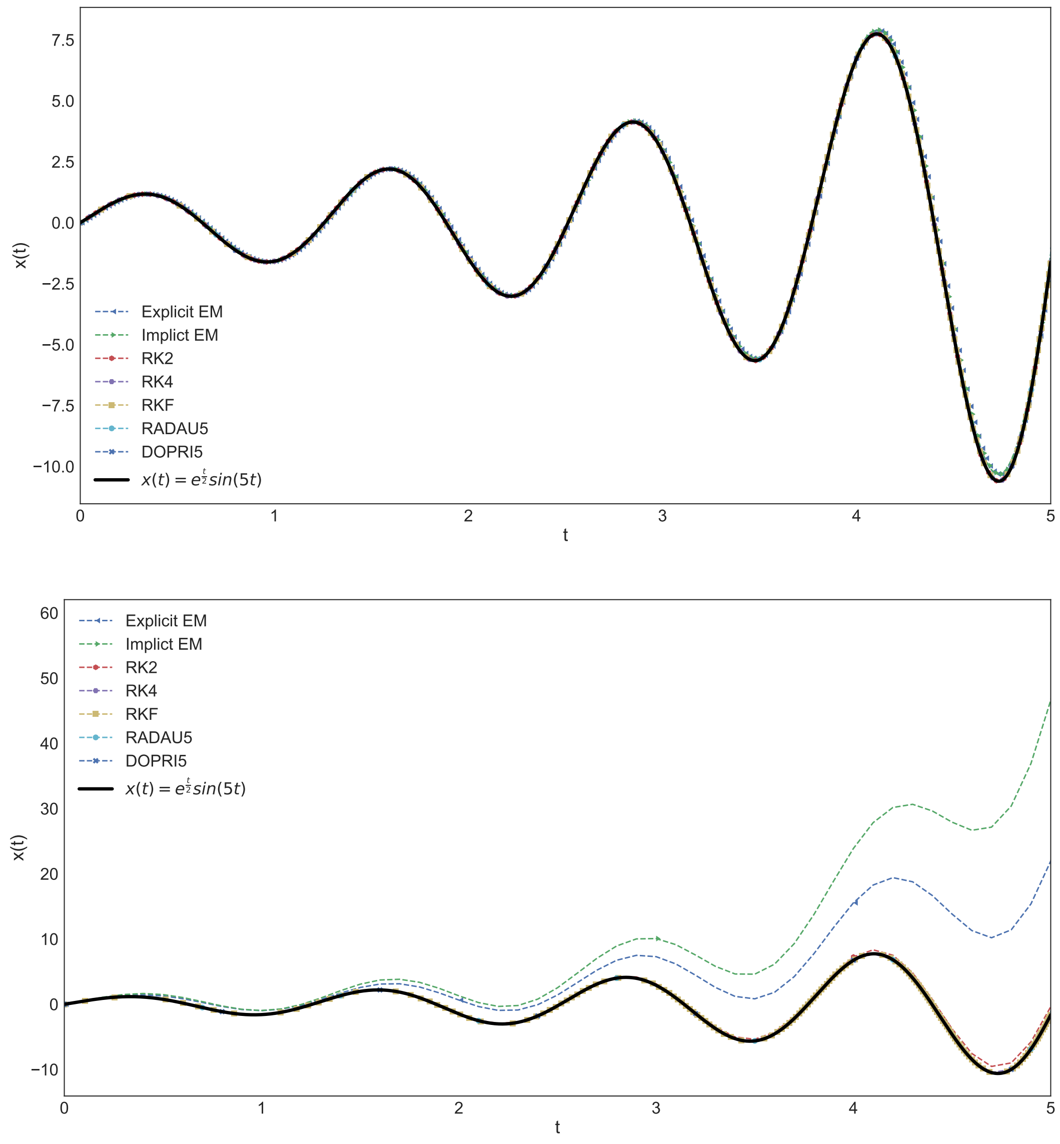


Fig. 1.3 Numerical solutions obtained by explicit EM, implict EM, RK2, RK4, RKF, DOPRI5, and RADAU5 to solve the differential equation $x'(t) = -0.5e^{\frac{t}{2}}\sin(5t) + 5e^{\frac{t}{2}}\cos(5t) + x(t)$ with initial condition $x(0) = 0$, whose exact solution is $x(t) = e^{\frac{t}{2}}sin(5t)$. The solutions obtained using $h = 0.001$ are shown in the top panel, while those obtained using $h = 0.1$ are depicted in the bottom panel. This simple example shows that fixed step-size methods (explicit EM, implicit EM, RK2 and RK4) require the selection of a proper h value—which is kept constant during the iterations of the methods—to well approximate the exact solution.

LSODA

Livermore Solver of Ordinary Differential Equations (LSODA) [341] is an ODE solver capable of automatically recognizing stiff and non-stiff systems, switching between the most appropriate integration family of methods: (*i*) the Adams-Moulton family [176] in the absence of stiffness, and (*ii*) the BDF otherwise. Both families are multi-step methods, so the current solution depends on more than one previous state. Initially, the problem is assumed to be non-stiff and during the integration some heuristics are exploited to evaluate the stiffness. If the problem becomes stiff, LSODA automatically switches to the BDF family. As in the case of implicit EM and Radau IIA family, LSODA exploits the Jacobian matrix of the system of ODEs, which must by provided by the user as a function (LSODA implementation is available in `FORTRAN`, `C` and `Python` programming languages). In order to control the performance and quality of the solution of the ODEs, LSODA has several functioning settings that can be provided by the user. Among them, the most important are the absolute and relative error tolerances (as in the case of RADAU5 and DOPRI5), and the maximum number of internal iterations allowed for a single integration step.

The Variable-coefficient ODE solver (VODE) [53] exploits the same integration method family of LSODA. Differently from LSODA, VODE does not switch between the two families of solvers during the integration, but it selects the most appropriate family at the beginning of the integration by exploiting some heuristics.

1.2.2 Stochastic simulation

Stochastic modeling approaches are suitable to represent biochemical systems when some molecular species occur in a low number of copies in the system Ω. In such a case, the simulated trajectories diverge from those generated by the deterministic simulation. Indeed, stochastic approaches allow for a deeper knowledge of the behavior of systems where intrinsic biological noise plays a fundamental role, such as those characterized by bistability. According to a specific probability distribution for the firing of reactions, a stochastic simulation reproduces one of the possible trajectories that can occur. Differently from ODE modeling, stochastic approaches assume that the state of the system is discrete, that is, it is represented by a vector of integer-valued numbers whose values correspond to the exact amount of molecules of the chemical species occurring in Ω.

In stochastic modeling and simulation, given a system Ω we would like to know the probability that Ω will be in any state $\mathbf{x}$ at time t starting from the initial condition $\mathbf{x}_0$ at time t_0. As described in Section 1.1.1, this probability is given by the CME and can be obtained by calculating every possible state of Ω. Since the number of possible states of Ω increases exponentially with the number of chemical species, the CME cannot be easily used to investigate complex (biochemical reaction) systems. As a matter of fact, as a consequence of the reactions firing [211], a specific differential equation exist for each possible state than can be reached, leading to the so-called "curse of dimensionality".

During the years, numerical algorithms based on matrix descriptions of the discrete-state Markov process [412] were proposed to solve the CME. Considering systems consisting in a lot of molecular species, which lead to a huge number or even infinite reachable states, the computational demand makes these methods unfeasible. In literature there exist also analytic algorithms that allow for solving the CME. Among them, it is worth mentioning those based on uniformization methods [192, 397, 488], finite state projection algorithms [58, 296], or the sliding window method [470].

Differently from these solutions, stochastic simulation algorithms generate trajectories of the underlying Markov process to provide a solution that is equivalent to the CME. Among this family of algorithms, in the next section we describe the Gillespie's Stochastic Simulation Algorithm [153, 154].

Gillespie's Stochastic Simulation Algorithm

Gillespie introduced the Stochastic Simulation Algorithm (SSA) [153, 154] that generates exact realizations of the CME of a biochemical system. Given a system Ω, SSA is capable of providing trajectories of the associated continuous time and discrete state space jump Markov process $\mathbf{x}$ of Ω. Notice that the CME determines the initial conditional density function of the system [155].

SSA works as follows: given the state $\mathbf{x}$ of the system, the reaction that will take place in the next time interval $[t, t+\tau)$ has to be chosen. To this aim, the probability of each reaction R_i to occur in the next infinitesimal time step $[t, t+dt)$ is evaluated: SSA exploits the joint probability density function $P(i, \tau|\mathbf{x}, t)$ of the two random variables i (i.e., the index of the next reaction) and τ (i.e, the time to the next reaction), given that the system is currently in state $\mathbf{x}$ at time t. Gillespie proved that $P(i, \tau|\mathbf{x}, t)$ can be formulated as

$$P(i, \tau|\mathbf{x}, t) = \alpha_i(\mathbf{x}) \exp(-\alpha_0(\mathbf{x})\tau),$$

where $\alpha_i(\mathbf{x}) = c_i \cdot d_i(\mathbf{x})$ is the so-called *propensity function* of the reaction R_i—where c_i is the stochastic constant associated with the reaction, and $d_i(\mathbf{x})$ is the number of distinct combinations of the reactant molecules in R_i occurring in state $\mathbf{x}$—and $\alpha_0(\mathbf{x}) = \sum_{i=1}^{M} \alpha_i(\mathbf{x})$. Then, SSA computes the time τ before a reaction takes place as:

$$\tau = \frac{1}{\alpha_0(\mathbf{x})} \ln\left(\frac{1}{\rho_1}\right),$$

where ρ_1 is a random number sampled in $[0,1]$ with a uniform probability. The reaction R_i to be actually executed is then chosen by taking the smallest integer in the set $1,\dots,M$ such that

$$\sum_{i\prime=1}^{i} \alpha_{i'}(\mathbf{x}) > \rho_2 \cdot \alpha_0(\mathbf{x}),$$

where ρ_2 is a second random number sampled in $[0,1]$ with a uniform probability.

This is the traditional formulation of SSA, called the Direct Method (DM). The First Reaction Method (FRM) is a variant of the DM introduced by Gillespie in which a putative time τ_i is calculated for each reaction R_i. Once all the putative times are known, the reaction that has the smallest putative time is applied. Even if they seem different, Gillespie proved that the DM and the FRM are equivalent [153].

The most expensive part of SSA consists in calculating the propensity function at each step. As a matter of fact, the only propensity functions that should be updated are those related to the reactions whose reactants were interested by a reaction fired in the previous simulation step. Gibson and Bruck introduced an approach based on the FRM, called the Next Reaction Method (NRM) [152], which exploits a dependency graph to update at each simulation step only the propensity functions whose reactants were interested in the previous step. Moreover, in the NRM the putative times are reused to reduce the computational time by avoiding the generation of random numbers, which is a computationally expensive task. Finally, the NRM takes advantage of optimized priority queues to store both the propensity functions and putative times in order to efficiently perform the updates.

The improvements proposed by the NRM were not enough to reduce the computational burden of SSA, especially in the case of biochemical systems characterized by a lot of reactions and chemical species. In order to deal with this issue, Gillespie proposed *τ-leaping* [157], which is an approximate but faster version of SSA. The major modification introduced in τ-leaping regards the reactions that can be applied at each step: in this approximate version *several* reactions can be applied during each step of the method, instead of a *single* reaction as is the case of SSA.

1.2.3 State-of-the-art

Among the CPU-based simulators of biochemical models, it is worth mentioning the COmplex PAthway SImulator (COPASI) [202], which integrates different algorithms, including LSODA and SSA, and is one of the most used tools in the community of Systems Biology. Since in this thesis we exploit High-Performance Computing (HPC) solutions to accelerate both deterministic and stochastic simulations, in this section we provide an overview of the existing simulators accelerated on Graphics Processing Units (GPUs). The existing tools can be classified with respect to two main concepts: the simulation granularity and the simulation type [308]. The first category determines how the threads are used: in coarse-grained simulation each thread corresponds to an independent simulation, while in fine-grained simulation the calculations of a single simulation are distributed over multiple threads. The second category determines the type of simulation: deterministic or stochastic.

Regarding deterministic simulation, Ackermann *et al.* [3] developed a GPU-accelerated simulator to execute massively parallel simulations of biological molecular networks. This methodology automatically converts a model, described using the Systems Biology Mark-up Language (SBML) standard [229], into a specific Compute Unified Device Architecture (CUDA) implementation of the EM (see Section 3.4.1 for further details about CUDA). The authors developed also a CPU version of the EM to test the efficiency of the proposed GPU simulator. The evaluation of this implementation on a Nvidia GeForce 9800 GX2 showed a speed-up between 28$\times$ and 63$\times$, compared to the execution on a CPU Xeon 2.66 GHz. In a similar vein, a simulator developed in CUDA (named cuda-sim), which exploits LSODA as ODE solver, was presented by Zhou *et al.* [491]. Notice that cuda-sim implements also a GPU-powered version of SSA. The cuda-sim simulator performs the so-called "just in time" (JIT) compilation (that is, the creation, compilation and linking at *run-time* of new source code) by converting an SBML model into CUDA code. With respect to the CPU implementation of LSODA contained in the numpy library of `Python`, cuda-sim achieved a 47$\times$ speed-up. Nobile *et al.* [306] presented another parallel simulator relying on the LSODA algorithm, named cupSODA, to speed-up the simultaneous execution of a large number of deterministic simulations. Differently from cuda-sim, cupSODA saves execution time by avoiding JIT compilation and by relying on a GPU-side parser. cupSODA achieved an acceleration up to 86$\times$ with respect to COPASI [202], used as reference CPU-based LSODA simulator. This relevant acceleration was obtained thanks to a meticulous optimization of the data structures and an intensive usage of the whole memory hierarchy on GPUs (described in detail in Section 3.4.1).

When stochastic simulations are taken into account, a problematic issue is the availability of GPU-side high-quality random numbers generators (RNGs). Since the CURAND library (see Section 3.4.1) was introduced with the fourth release of CUDA, early GPU implementations required the development of custom kernels for RNGs. This problem was faced for the CUDA version of SSA developed by Li and Petzold [259], who implemented the Mersenne Twister RNG [278], achieving a 50$\times$ speed-up with respect to a common single-threaded CPU implementation of SSA. Sumiyoshi *et al.* [419] extended this methodology by performing both coarse-grain and fine-grain parallelization: the former allows for multiple simultaneous stochastic simulations of a model, while the latter is achieved by distributing over multiple threads the calculations related to the model reactions. This version of SSA achieved a 130× speed-up with respect to the sequential simulation on the host computer. The τ-leaping algorithm allows for a faster generation of the dynamics of stochastic models with respect to SSA, by properly calculating longer simulation steps [159, 64]. Komarov *et al.* [235] proposed a GPU-powered fine-grained τ-leaping implementation, which was shown to be efficient in the case of extremely large (synthetic) biochemical networks (i.e., characterized by more than 10^5 reactions). Nobile *et al.* [307] then proposed cuTauLeaping, a GPU-powered coarse-grained implementation of the optimized version of τ-leaping proposed by Cao *et al.* [64]. Thanks to the optimization of data structures in low-latency memories and to the splitting of the algorithm into multiple phases corresponding to lightweight CUDA kernels, cuTauLeaping was up to three orders of magnitude faster on a GeForce GTX 590 GPU than the CPU-based implementation of τ-leaping contained in COPASI, executed on a CPU Intel Core i7-2600 3.4 GHz. A mixed approach of fine- and coarse-grained accelerations is GPU-ODM [234], which is based on a variant of SSA called the Optimized DM.

The simulators proposed in this thesis—i.e., LASSIE (LArge-Scale SImulator) [425] and FiCoS, together with a coarse-grained implementation of SSA by means of Many Integrated Core coprocessors [427]—will be extensively described and analyzed in Chapter 4.

1.3 Parameter Estimation in Systems Biology

The design and development of efficient methods to analyze the functioning of cellular systems represent one of the main goals of Computational Systems Biology. The emergent behavior is a system-level property due to the complex interactions among lots of molecular species. The analyses of these complex interactions cannot be performed by only relying on classic experimental research based on *in vivo* experiments. As described in the previous sections, mathematical modeling along with simulation techniques have become valuable tools that are capable of describing and understanding the molecular mechanisms taking place in cellular processes. However, mathematical (mechanism-based) models require proper quantitative parameterizations to perform *in silico* simulations of the system dynamics in physiological or perturbed conditions [84]. These chemico-physical parameters (e.g., reaction rates) are fundamental for biochemical modeling since they drive the emergent behavior of the system. As a matter of fact, sometimes even a small change in the parameters values can dramatically change the output dynamics. Figure 1.4 shows how small changes in the initial value of the parameters lead to completely different dynamics in the case of a simple model like the Brusselator (see Appendix A.1).

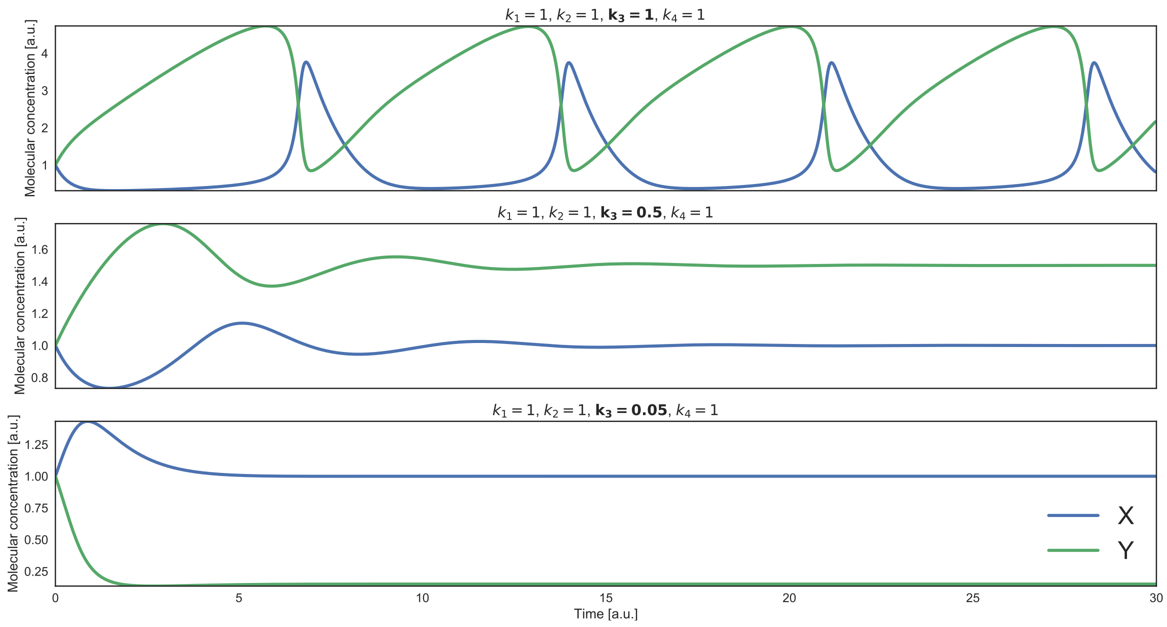


Fig. 1.4 Example of different emergent behaviors obtained by varying the kinetic parameters. The Brusselator is a simple and theoretical model for an autocatalytic reaction, characterized by four reactions and five chemical species (see Appendix A.1 for further details). The top panel shows the dynamics of the species X and Y obtained using the following kinetic parameters: $k_1 = 1$, $k_2 = 1$, $k_3 = 1$, $k_4 = 1$. By setting k_3 equal to 0.5 the resulting dynamics are completely different (middle panel). By varying k_3 from 0.5 to 0.05 the oscillations are no longer present (bottom panel). This simple example shows that the parameters play a fundamental role, driving the emergent behavior of the system under investigation.

The lack of knowledge of kinetic parameters limits the effectiveness of mathematical modeling and *in silico* simulations. These parameters are generally hard or even impossible to measure directly by means of classic *in vivo* experiments, leading to the definition of PE problem [84, 356]. The main goal of the PE problem is the inference of the unknown values of the model parameters. In this thesis, this problem will correspond to determine the unknown values of the constants associated with the set of reactions $\mathcal{R} = \{R_1, \dots, R_M\}$ in an RBM. Notice that, when some prior knowledge about a number of kinetic constants is available, the PE task can be performed by considering only a subset of the reactions $\mathcal{R}' = \{R_1, \dots, R_D\} \subseteq \mathcal{R}$, with $D \leq M$. The estimation process is carried out by exploiting the availability of some other experimental data (related, for instance, to time-series amounts of some molecular species occurring in the system), which can be measured by means of classic laboratory experiments and protocols.

1.3.1 State-of-the-art

Several PE methodologies was proposed to deal with the lack of data when both deterministic and stochastic simulation approaches are taken into account. Generally, these methodologies rely on approximation strategies [263, 356, 368], probabilistic methods [300, 347, 460], global optimization [43, 118, 294], or a combination of these approaches [487].

Probabilistic methods and approximation strategies have been applied especially for the PE of stochastic biological systems. Since they require a huge amount of simulations to obtain reliable results, becoming very time consuming, they are unfeasible in the case of large-scale systems.

The traditional methods based on the Gradient Descend or local search strategies (described in Section 2.1.2) are also not suitable to address the PE problems because (*i*) the fitness landscape is non-linear, non-convex and multi-modal, thus the probability that these methods converge to a local minimum is high (also by applying multi-start strategies [285]); (*ii*) when stochastic simulations are considered, the fitness landscape is rugged due to the stochastic fluctuations, which generally produce a lot of local minima increasing the difficulty of the optimization problem.

In order to overcome the limitations afflicting the methods based on the Gradient Descend, global optimization techniques (see, e.g., Sections 2.2 and 2.3) can be applied. In [285], the authors showed that Simulated Annealing (see Section 2.1.3) was able to outperform local search approaches for the PE of the mechanism of irreversible inhibition of HIV proteinase. The main drawback of the proposed approach is represented by the computational time required to perform the PE of this simple model. Mendes [284] applied different optimization methods to deal with the PE of a large three-step pathway characterized by 36 kinetic parameters to estimate. The results indicate that Evolutionary Programming is the best choice since it found a set of parameters that allows for obtaining simulated dynamics overlapping the target of the analyzed system, while the other tested methods failed in finding good parameterizations. A benchmark model consisting in 36 parameters have been used as case study in [294]. The authors tested several global optimization methods, showing that the most suitable approach to solve the PE problem, among those considered in this work, is the Evolution Strategy using Stochastic Ranking (SRES) [369]. This effective method uses a stochastic ranking as the constraint handling technique, which exploits the bubble-sort algorithm, to automatically adjust the balance between the fitness and penalty functions during the evolutionary search. Also in this case, the main drawback is represented by the time required to perform a computation, burdened by the sorting algorithm. Dräger *et al.* [118] performed an empirical comparison of six alternative models of valine and leucine bio-synthesis in *C. glutamicum*, characterized by different levels of complexity and considering different kinetic equations, resulting in up to 59 parameters to be estimated. The authors reported that, when the settings of Particle Swarm Optimization (PSO) (see Section 2.3.4) are carefully tuned, it results the most efficient algorithm for the PE problem.

Although global optimization methods require the execution of a non negligible number of fitness evaluations, they are the best candidates to solve the PE problem. Moreover, the computation of the fitness functions can be parallelized by using some HPC architecture, especially GPUs (see Chapter 5 for further details).

1.4 The Haplotype Assembly problem in Genome Analysis

Somatic human cells are diploids, that is, they contain 22 pairs of homologous chromosomes and a pair of sex chromosomes, one copy inherited from each parent. In order to fully characterize the genome of an individual, the reconstruction of the two distinct haplotypes is essential [254]. The inference of the full haplotype information is known as haplotyping, which consists in assigning all heterozygous Single Nucleotide Polymorphisms (SNPs) to exactly one of the two chromosome copies. SNPs are one of the most studied genetic variations, since they play a fundamental role in many medical applications (e.g., drug-design, disease susceptibility studies). This information can be valuable in several contexts, including linkage analysis, association studies, population genetics, and clinical genetics [404].

The advent of second-generation sequencing technologies revolutionized the field of genomics, enabling a more complete view and understanding of the genome of different species. However, despite their great contribution to the field, the data produced by these technologies are still unsuitable for several applications, including Haplotype Assembly. The short length of the reads produced by second-generation sequencing technologies might be not long enough to span over a relevant number of SNP positions, leading to the reconstruction of short haplotype blocks [489, 99] and ultimately hindering the possibility of reconstructing the full haplotypes. In recent years, however, a third-generation of sequencing technologies was developed and paved the way to the production of sequencing data characterized by reads covering hundreds of kilobases, thus able to span different SNP loci at once [358, 363, 212]. Unfortunately, the increase in length comes at the cost of a decrease in the accuracy of the reads, compared to the short and precise ones produced by second-generation sequencing technologies, such as NovaSeq (Illumina Inc., San Diego, CA, USA) [350]. In order to compensate for this inadequacy, there is a need to increase read coverage. Formally, the coverage of a sequencing experiment is the average number of times that each nucleotide is expected to be covered by a read. This value is given by the following relationship:

$$\text{cov} = (L \cdot N)/G, \tag{1.22}$$

where cov stands for the coverage, L for the read length, N for the number of reads and G for the length of the haploid region of the genome on which the reads are mapped [245]. Equation 1.22 shows that longer reads or a higher amount of reads are needed to increase the coverage. In practice, an average coverage higher than 30× is the *de facto* standard for accurate SNP detection [398].

1.4.1 Current trends in sequencing experiments

The first studies regarding Single Nucleotide Variations (SNVs) showed that all homozygous SNVs can be effectively detected by using sequencing experiments with average coverage equal to 15×, while a higher coverage (i.e., 33× on average) is necessary to discover the same proportion of heterozygous SNVs [398]. In the latest years, a sequencing coverage of 35× is the *de facto* standard for SNV as well as insert and deletion (InDel) detection, even if Ajay *et al.* [6] suggested that an average coverage of 50× is required to allow for reliable calling of SNVs and small InDel analyses.

Whole Genome Sequencing (WGS) is becoming more and more important in many applications, ranging from SNV to Copy Number Variation (CNV) detection. Depending on the aims of the proposed study, the typical coverage for WGS experiments varies from 1×-8× for CNVs to 60× for InDel analyses [149, 13]. Due to the high cost of the WGS experiments, a coverage equal to 15× is the most common.

Considering Whole Exome Sequencing (WES) studies, a greater average read depth is mandatory to achieve the same breadth of coverage of WGS experiments, requiring an 80× average depth to cover 89.6%-96.8% of the target bases [398].

De novo assembly requires a sequencing depth between 38× and 56× [149, 398]. As shown in [355], the minimum sequencing coverage from technologies producing short reads should be equal to 29×. In [76], the authors proposed a comparative study about *de novo* assembly, showing that long reads can be applied to this purpose; however, exploiting long reads alone with a coverage below 35× is not sufficient to produce satisfying results. This study suggested that, in the case of low coverage, hybrid assembly methods are the best choices.

During the latest years, single-cell RNA sequencing (scRNA-seq) experiments gained ground since they provide the expression profile of individual cells, allowing for revealing complex and rare cell populations [204], which are fundamental for the characterization of the sub-population structure of a heterogeneous cell population. Downstream analyses are applied to uncover regulatory relationships between genes as well as track the trajectories of distinct cell lineages in development. In this field, the coverage of the underlying sequencing experiments is generally defined as the number of reads per cell. Depending on the scRNA-seq approach and the purpose of the study, different read depth (per cell) are used, varying from 10^4 to 10^6 [185]. For instance, the different cell types of a population can be classified by exploiting a sequencing depth of $5 \cdot 10^4$ reads per cell, which allows for accurate and reliable results [414, 185].

Finally, for SNP and SNV detection as well as small InDel identifications, the minimum coverage should be equal to $30\times$ [398].

1.4.2 State-of-the-art

Several computational Haplotype Assembly approaches for human genome phasing have been proposed in literature [83]. Most of these methods solve the NP-hard Minimum Error Correction (MEC) problem, which aims at inferring the haplotype pair that yields two disjoint sets of the sequencing reads characterized by the minimum number of SNP values to be corrected [457]. An additional variant of MEC exists, called weighted MEC (wMEC) [169], which takes into account also the information concerning the quality of the reads. Figure 1.5 shows a “phylogenetic tree”-like diagram of the existing haplotyping methods, which are briefly described in what follows.

Beagle [55] is one of the earliest heuristic approaches based on Hidden Markov Models (HMMs). Considering the genotype information of an individual, Beagle finds the most likely haplotype pair among different possible haplotype solutions. It has a quadratic computational complexity with respect to the input data. SHAPEIT [110] starts from genotyping data from a population and, given the genotype data of an individual, exploits an HMM-based approach to estimate the haplotype pair. The population data are used to apply constraints on the graph, which denotes all possible haplotypes compatible with the input data, in order to determine the haplotype of that individual. At each iteration, SHAPEIT has a linear complexity with respect to the number of haplotypes. Eagle2 [266] is a phasing algorithm that exploits the Burrows-Wheeler transform to encode the information from large external reference panels. It relies on an HMM to explore only the most relevant phase paths among all possible paths. The authors showed that Eagle is 20 times faster than SHAPEIT [110]. HapCUT [28] leverages sequencing data (i.e., the entire set of reads is considered) instead of population genotypes. It infers the haplotype pair of an individual by partitioning the set of reads solving the MEC problem. The MEC problem is reduced to the max-cut problem, which is greedily solved over the graph representation of the input instance. HapCUT2 [124] is a

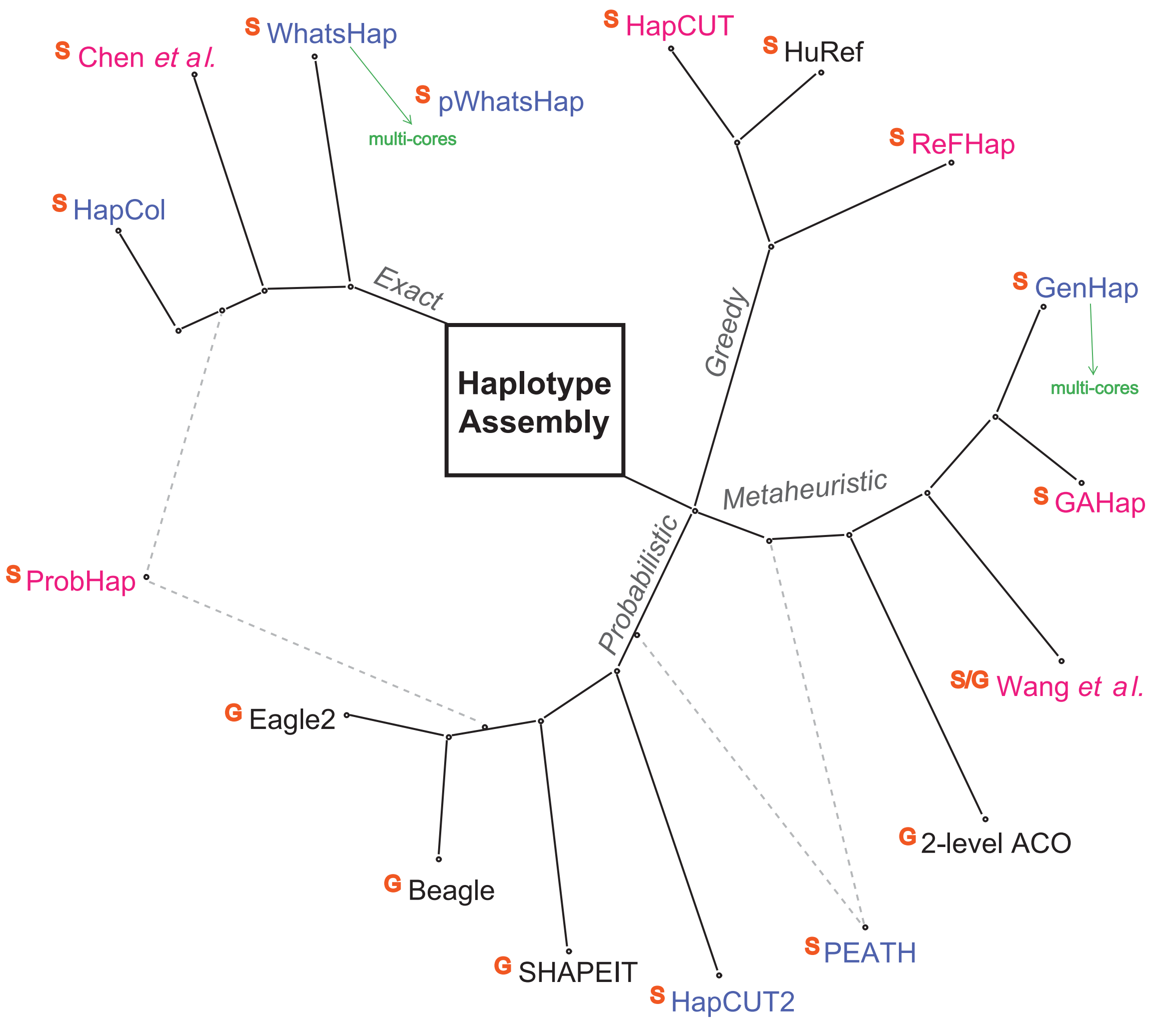


Fig. 1.5 The "phylogeny" of haplotyping methods. Over the past few years, the repertoire of tools for haplotyping has rapidly expanded. A "phylogenetic tree"-like diagram is used here to depict the division of the algorithms in four different classes, namely: exact, greedy, probabilistic, metaheuristic. Hybrid methods are connected with dashed lines to the implemented multiple computational techniques. The orange superscript denotes the analyzed data: sequencing (S) and genotyping (G). Methods that solve either the MEC or the wMEC problem are denoted with blue or magenta, respectively. Finally, the Haplotype Assembly methods that exploit HPC are highlighted with a green arrow directed to the used computational resources.

recent heuristic approach that exploits a haplotype likelihood model for the sequencing reads. A partial likelihood function is used to evaluate the likelihood of a subset of the fragments. Differently from its previous version (HapCUT [28]), which is based on a max-cut algorithm, HapCUT2 optimizes the likelihood to find a max-cut in graph representation of the input instance.

ProbHap [239] relies on an exact likelihood optimization technique to solve a generalized version of the MEC problem. It exploits a dynamic programming algorithm capable of exactly optimizing a likelihood function, which is specified by a probabilistic graphical model that generalizes the MEC problem. ProbHap can handle long reads coverage values up to $20\times$, which is not appropriate for higher coverage short-read datasets; on the other hand, it works better with very long reads at a relatively shallow coverage ($\leq 12\times$).

ReFHap [121] is based on a heuristic algorithm to find the max-cut. ReFHap solves the Maximum Fragments Cut (MFC) problem instead of the classic MEC problem. The max-cut problem is reduced to the MFC problem, which is addressed using a greedy approach. HuRef [254] is a heuristic approach that aims at inferring the heterozygous variants of an individual. It is based on a greedy algorithm that iteratively refines the initial partial haplotype solutions. The authors leveraged this Haplotype Assembly approach to study non-SNP genetic alterations considering the diploid nature of the human genome.

Chen *et al.* (2013) [81] proposed an exact approach for the MEC problem using an integer Linear Programming solver. First, the fragment matrix is decomposed into small independent sub-matrices. Each of these sub-matrices is used to define an integer Linear Programming problem that is then exactly solved. WhatsHap [333] is an exact method relying on a dynamic programming algorithm used to solve wMEC. It implements a fixed parameter tractable algorithm [191, 51], where the fixed parameter is the maximum coverage of the input instance, to deal with the NP-hardness of the wMEC problem, leveraging the long-range information of long reads. This method does not assume the all-heterozygosity of the phased positions, however, it can deal only with datasets of limited coverage up to $\sim 20\times$. pWhatsHap [52] is an efficient version of WhatsHap [333], which was designed to leverage multi-core architectures in order to obtain a relevant reduction of the execution time required by WhatsHap. The proposed implementation exploits the physical shared memory of the underlying architecture to avoid data communication among threads. HapCol [342] implements a dynamic programming algorithm to solve an alternative version of the wMEC problem, called k-MEC, which is used to take into account the distribution of sequencing errors of future-generation technologies. In this strategy, the number of corrections per column is bounded by the parameter k. No all-heterozygous assumption is required, but it can only deal with instances of relatively small coverages up to $\sim 25-30\times$.

Two-Level ACO [36] is based on the Ant Colony Optimization (ACO) technique (see Section 2.3.1), which is a metaheuristic designed to deal with combinatorial problems on graphs generated starting from the genotyping data given as input. This approach is based on the pure parsimony criterion to find the smallest set of distinct haplotypes that solves the Haplotype Assembly problem. Probabilistic Evolutionary Algorithm with Toggling for Haplotyping (PEATH) [298] is based on the Estimation of Distribution Algorithm (see Section 2.2.4), which is exploited to deal with noisy sequencing reads, aiming at inferring one haplotype, under the all-heterozygous assumption. The method proposed by Wang *et al.* (2005) [457] relies on Genetic Algorithms (GAs), which are extensively explained in Section 2.2.5, to address an extended version of the MEC problem, called MEC with Genotype Information (MEC/GI), which also considers genotyping data during the SNP correction process. GAHap [459] uses GAs to infer the haplotype pair of an individual working on nucleotide strings. During the optimization, GAHap solves the MEC problem by using a fitness function based on a majority rule that takes into account the allele frequencies. The results shown in [459] are limited to a coverage up to $10\times$ and a haplotype length equal to 700. No all-heterozygous assumption is required.

In Chapter 6 we will explain in details GenHap [433], which is a novel computational method based on GAs, designed to deal with the computational hardness of the Haplotype Assembly problem.

1.5 Medical image enhancement and segmentation

Medical imaging systems often require the application of image enhancement techniques to help physicians in anomaly/abnormality detection and diagnosis, as well as to improve the quality of images that undergo automated image processing. Appropriate image pre-processing steps can improve the result accuracy achieved by computer-assisted segmentation methods. Among the low-level intensity-based Pattern Recognition techniques, which are widely adopted in scenarios with real-time constraints, the simplest unsupervised image segmentation algorithm is global thresholding, which essentially reduces to a pixel classification problem [165]. In particular, image binarization classifies the input pictorial data into exactly two classes (i.e., foreground and background), given a threshold intensity value [365]. This global threshold value is efficiently computed by operating on the image histogram alone. Unfortunately, adaptive thresholding techniques for two-class segmentation work properly only for images characterized by bimodal histograms [477]. Therefore, in the case of images with a bimodal intensity distribution, image binarization techniques are able to classify the input pictorial data into two classes. In practice, different types of regions in an image could overlap, thus affecting the bimodality conditions of the gray level histogram, where the histogram modes semantically correspond to different types of regions. Image pre-processing can definitely improve the result accuracy achieved by computer-assisted segmentation methods [365], by sharpening the peaks of the two sub-distributions, so that the resulting histogram is characterized by a stronger bimodality, even in the case of blurred region contours and of the related Mach band effect pertaining to edge-detection in the human visual system [97, 238]. As a way of example, in radiology this phenomenon is accentuated in edges of adjacent regions that slightly differ in terms of gray level intensities [353].

No existing pre-processing technique addresses the issues related to medical image enhancement for subsequent binarization by using adaptive thresholding [477]. Literature methods may be inadequate when dealing with low-contrast images [257], producing false edges and under-/over-segmentation when input images are affected by noise, as in the case of Magnetic Resonance Imaging (MRI) data [144], which represents the leading modality for the imaging of soft-tissues in current medical practice, with particular relevance in cancer imaging, allowing for high-contrast between the tumors and the surrounding tissues [128]. Unfortunately, MRI data are affected by acquisition noise [400] and are also prone to imaging artifacts, mainly caused by magnetic susceptibility and large intensity inhomogeneities of the principal field (i.e., streaking or shadowing artifacts [402]), especially in the latest MRI acquisition devices with high magnetic field intensity.

1.5.1 State-of-the-art

Most of the existing enhancement techniques are empirical or heuristic methods—strongly related to a particular type of images—which generally aim at improving the contrast level of images degraded during the acquisition process [79]. As a matter of fact, finding the best gray level mapping that adaptively enhances each different input image can be considered an optimization problem [334, 117]. Unfortunately, no unifying theory employing a standardized image quality measure is currently available to define a general criterion for image enhancement [297]. In addition, in the case of medical imaging, techniques tailored on specific tasks are necessary to achieve a significant enhancement and, in general, interactive procedures involving considerable human effort are needed to obtain satisfactory results.

In order to achieve objective and reproducible measurements conveying clinically useful information, operator-dependence should be minimized by means of automated methods. Point-wise operations in the spatial (pixel) domain, representing the simplest form of image processing, are effective solutions since efficiency requirements have also to be met. In the case of image enhancement, they re-map each input gray level into a certain output gray level, according to a global transformation [165]. Thus, such kind of techniques treat images as a whole, without considering specific features of different regions, or selectively distinguishing between a collection of contrast enhancement degrees or settings [297]. Histogram Equalization (HE) is the most common global image enhancement technique, whose aim is to uniformly redistribute the input gray level values according to the cumulative density function of its histogram [165, 180]. Unfortunately, HE does not take into account the image mean intensity [80], which is subject to a significant change during the equalization process by invariably shifting the output mean brightness to the middle gray level, regardless of the mean gray level in the input image [143]. Consequently, HE is not able to preserve the input mean brightness, possibly suffering from over-enhancement, and giving rise to artifacts such as the so-called washed-out effect [80]. This global transformation method applies contrast stretching just on gray levels with the highest frequencies, causing a significant contrast loss concerning the gray levels characterized by low frequencies in the input histogram [227]. Bi-Histogram Equalization (Bi-HE), which is a refined version of the traditional HE, was proposed to overcome the limitations related to input mean brightness preservation, mainly caused by histogram flattening [227]. Firstly, Bi-HE splits the original histogram into two sub-histograms according to the global mean of the original image; afterwards, the sub-histograms are independently processed by applying the standard HE method to each of them.

In addition to HE, which automatically yields an image with a uniform histogram, it is possible to explicitly specify the desired shape of the output histogram. This method, named Histogram Specification (HS), aims at matching the histogram of the gray level intensities of the input image against a desired histogram [165]. Unfortunately, this approach cannot be applied in the case of image datasets characterized by heterogeneous gray level distributions, since the histogram to be matched should be defined either *a priori* for all the images in the dataset, or interactively for each processed image, by separating and shaping the two underlying sub-distributions [473].

Other traditional global gray level transformations generally used for contrast stretching are formalized as transformation functions of the form $s = \mathcal{T}(r)$, where $\mathcal{T}(\cdot)$ maps an input intensity value r into an output intensity value s [165]. Power-law transformation—also called Gamma Transformation (GT)—is a non-linear operation of the form $\mathcal{T}(r) = cr^{\gamma}$, where typically $c = 1$. For instance, when the image is predominantly dark, an expansion of the intensity levels is desirable. In such a case, GT with $\gamma < 1$ yields a brighter image by increasing the number of hyper-intense pixels; on the contrary, by using $\gamma > 1$, the GT converts the input gray-scale range into a darker one, by increasing the occurrences of darker pixels. Obviously, the value of γ strongly depends on the (medical) application. Accordingly, logarithmic and anti-logarithmic transformations make an image much brighter and darker, respectively. Unfortunately, for medical images characterized by low-contrast and weak edges at adjacent tissue boundaries, GT may result in merely brighter or darker images, leading to difficulties in the visualization and discrimination of different tissues. Therefore, to adequately enhance contrast, the two different behaviors of GT—corresponding to values $\gamma > 1$ and $\gamma < 1$—should be combined for contextually decreasing the darker pixel gray values and increasing the brighter pixel gray values. This results in a significant improvement of the contrast, by enhancing the edges thanks to the increased gradient magnitude of the image [144]. This kind of contrast stretching can be achieved by using a Sigmoid intensity Transformation (ST), which darkens a wide range of hypo-intense gray levels and brightens a wide range of hyper-intense gray levels [165]. Such an operation indirectly increases the difference between low and high intensity values, resulting in the overall contrast enhancement of the image [144].

The complexity of the enhancement criteria to be met (i.e., the effective contrast stretching combined with image detail preserving) leads to the application of global search metaheuristics that allow for coping with several constraints, which are not generally tractable by means of traditional exhaustive computational approaches [334, 297, 325]. Evolutionary methods have been widely adopted in the image enhancement domain to find the optimal enhancement kernel [297], sequence of filters [233], or input-output mapping transformation [381, 65]. Recently, [188] proposed a GA-based method that efficiently encodes the histogram by means of the non-zero intensity levels, by employing genetic operators that directly process images to increase the visible details and contrast of low illumination regions, especially in the case of high dynamic ranges. The authors argued that this method yields “natural-looking” images, considering the visual appearance.

Regarding other evolutionary computation approaches, Genetic Programming (GP) [236] was shown to be a powerful framework to select and combine existing algorithms in the most suitable way. Differently to GAs, GP evolves a population of functions, or more generally, computer programs to solve a computational task. The solutions in the computer program space can be represented as trees, lines of code, expressions in prefix or postfix notations as well as strings of variable length [69]. For instance, [49] tackled the video change detection problem (among the frames of video streams) by combining existing algorithms *via* different GP solutions exploiting several fusion schemes. The fitness function was composed of different performance measures regarding change detection evaluation. For what concerns the application of GP in image enhancement, [343] proposed an approach to yield optimally pseudo-colored images for visualization purposes, aiming at combining multiple gray-scale images (e.g., time-varying images, multi-modal medical images, and multi-band satellite images) into a single pseudo-color image. This approach relies on user interactions to determine which candidate solution should be the winner in *tournament* selection, so it does not explicitly require a fitness function. As case studies, a pair of brain MRI sequences were fused as well as the motion of the heart on echocardiographic images was synthesized into a single pseudo-color image. Other works exploited Swarm Intelligence techniques. The approach presented in [392], called Multi-Objective Histogram Equalization, uses PSO [225] to enhance the contrast and preserve the brightness at the same time. The work presented in [117] employed the same encoding of candidate solutions and histogram mapping strategy described in [188], within an optimization strategy based on the Artificial Bee Colony (ABC) algorithm [223]. However, since ABC natively works in a continuous space, while a discrete representation is used for the solutions (i.e., gray-level mapping), a discretization step is mandatory in the correction operation during the search phase. An alternative approach using the ABC algorithm for image contrast enhancement was proposed in [79], wherein the optimal values for the parameters of a parametric image transformation, namely the Incomplete Beta Function, are estimated. Differently to the work described in [117], the optimization procedure is carried out in a continuous search space. Finally, multi objective Bat Optimization and a neuron-based model of Dynamic Stochastic Resonance were combined in [399] for the enhancement of brain MR images.

In Chapter 7 we will describe a novel image enhancement technique based on GAs, called MedGA [379], specifically aimed at strengthening the sub-distributions in medical images with an underlying bimodal histogram of the gray level intensities.

Chapter 2

Optimization techniques

An optimization problem is defined as the problem of finding the optimal solution by means of mathematical or computational methodologies, among all feasible solutions in a D-dimensional search space that is generally bounded (i.e., a set of given constraints bounds the search space). The most common optimization problems are minimization (maximization) problems, in which the solution $\mathbf{x}^* \in \mathbb{R}^D$ minimizing (or maximizing) an objective function $\mathrm{F} : \mathbb{R}^D \to \mathbb{R}$ must be identified. All the optimization problems considered in this thesis are expressed as minimization problems, where $\mathbf{x}^*$ is the optimal solution if and only $\mathrm{F}(\mathbf{x}^*) < \mathrm{F}(\mathbf{x}')$, $\forall \mathbf{x}' \neq \mathbf{x}^*$. Notice that any maximization problem can be easily formulated as a minimization problem by inverting the sign of the objective function, and viceversa.

The most naïve approach consists in enumerating all the feasible solutions, evaluating them by means of the objective function and ranking them according to the calculated values. This method cannot be applied due to the size of the search space, which may be too large or even infinite. Several methods have been proposed to efficiently explore the search space in order to find the best solution of the given objective function.

In this chapter, the classic optimization techniques will be presented, then Evolutionary Computation (EC) and Swarm Intelligence (SI) will be introduced and some examples of algorithms belonging to these fields will be explained. We highlight that Computational Intelligence is the field of research involving Neural Networks, Fuzzy Systems, EC and SI [120]. All these disciplines are based on practical adaptation, self-organization concepts and algorithms that can promote actions in complex environments. In this thesis, EC and SI methods have been taken into account to solve complex real world problems.

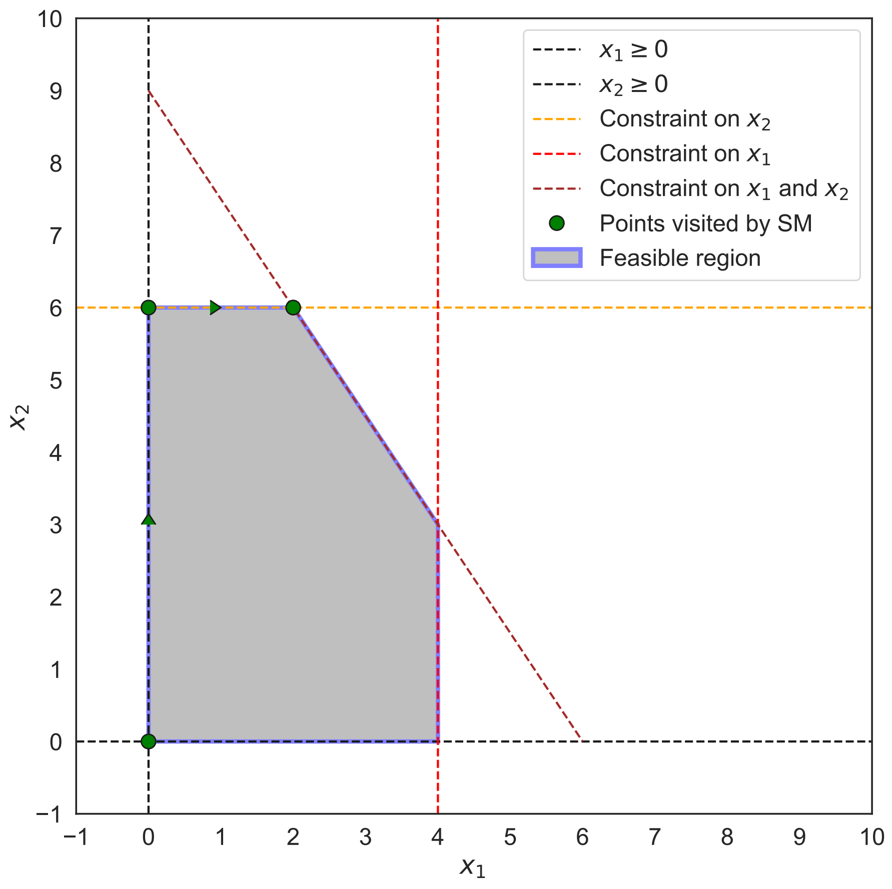


Fig. 2.1 Example of the execution of the SM maximizing the objective function described in Equation 2.1. The blue lines represent the 2-polytope defined by the 5 constraints (Equation 2.2), that is, 3 inequations and the non-negativity conditions of the variables x_1 and x_2. The 3 inequations are represented by the orange, red and brown dashed lines, while the non-negativity conditions by using the black dashed lines. The gray polygon indicates the feasible region obtained considering the aforementioned constraints. SM starts selecting an initial vertex as candidate solution $\mathbf{x}^*$ of the problem (here $\mathbf{x}^* = (0,0)$). The algorithm moves across the vertices (green dots), by following the edges of the polytope improving the objective function (green arrows). When no improving directions are found, the SM stops. By so doing, the last visited vertex corresponds to the global optimum (here $\mathbf{x}^* = (2,6)$).

2.1 Classic optimization techniques

2.1.1 Simplex Method

The Simplex Method (SM) [101] is probably the most known traditional technique used to solve Linear Programming (LP) problems where both the objective function and the constraints are linear. In SM, as shown in Figure 2.1, the space of the feasible solutions is described by means of a convex D-polytope, which is a polytope characterized by a convex set of points in the D-dimensional space generated by the linear constraints.

SM is an exact iterative algorithm in which, given a initial vertex $\mathbf{x}^* \in \mathbb{R}^D$ of the D-polytope, the objective function is evaluated on the vertices of the set V connected to $\mathbf{x}^*$ by means of an edge. Considering a maximization problem, the vertex in V characterized by the highest value (i.e., $\mathrm{F}(\mathbf{x}') > \mathrm{F}(\mathbf{x}^*)$ and $\mathrm{F}(\mathbf{x}') > \mathrm{F}(\mathbf{x}) \; \forall \, \mathbf{x}', \mathbf{x} \in \mathrm{V}$ such that $\mathbf{x}' \neq \mathbf{x}$) is selected as the new optimal solution $\mathbf{x}^*$ and the process is repeated. When no adjacent vertex improving the objective function is found (i.e., $\mathrm{F}(\mathbf{x}') \leq \mathrm{F}(\mathbf{x}^*)$, $\forall \, \mathbf{x}' \in \mathrm{V}$), the SM stops. Figure 2.1 depicts the execution of the SM maximizing the following objective function:

$$\text{maximizing} \quad 3x_1 + 5x_2, \tag{2.1}$$

with 5 constraints:

$$\begin{cases} x_1 & \leq 4 \\ 2x_2 & \leq 12 \\ 3x_1 + 2x_2 & \leq 18 \\ x_1 & \geq 0 \\ x_2 & \geq 0 \end{cases}, \tag{2.2}$$

with global optimum in $\mathbf{x}^* = (2, 6)$.

The described SM requires that the feasible region is convex and both the constraints and the objective function are linear. Moreover, in [231] the authors showed that for some classes of optimization problems SM has an exponential complexity in the worst case. Since many real world problems are strongly non-linear and non-convex (e.g., the Parameter Estimation of biological systems shown in Chapter 5), SM cannot be applied. Nevertheless, SM is used in some Systems Biology applications, such as the Flux Balance Analysis (FBA) [324], in which the vector of reaction fluxes in a biochemical system must be optimized. A classic FBA problem is represented by the maximization of some products of the system (e.g., ATP [351] or biomass [57]), where the constraints are represented by some biophysical limits (e.g., fluxes limitations or mass balance). Since a steady-state assumption is taken into account, the optimization of these fluxes can be stated as a LP problem and can be effectively solved by exploiting the SM.

2.1.2 Gradient Descent

As discussed above, the SM can be exploited when the region of feasible solutions is convex, the constraints as well as the objective function are linear. These assumptions limit the

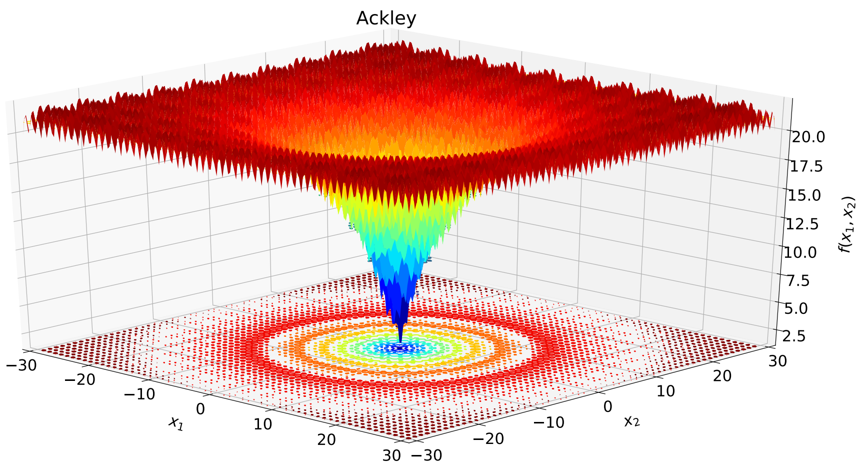


Fig. 2.2 The Ackley function is a classic benchmark function used to evaluate the performance of the optimization algorithms. It has a global optimum in $\mathbf{0}$ and several local optima. A gradient descent method is not able to converge to the global optimum, unless the initial solution is sampled close to $\mathbf{0}$.

applicability of SM in several real-world applications that are typically strongly non-linear. Gradient Descent (GD) [136] can be used to solve problems characterized by non-convex search spaces or non-linear objective functions. Unfortunately, the GD-based methodologies require that the objective functions is differentiable. Starting from an initial optimal solution $\mathbf{x}^*$—chosen randomly, using some *a priori* distribution or taking advantage of some available domain knowledge—GD exploits the slope of the objective function to search a better solution. To exploit the slope of the objective function, the gradient of the function is calculated in $\mathbf{x}^*$ and according to the gradient direction, a new optimal solution is determined as:

$$\mathbf{x}^* = \mathbf{x}^* - \Delta \cdot \nabla \mathrm{F}(\mathbf{x}^*), \tag{2.3}$$

where $\Delta \in \mathbb{R}^+$ is the so-called *step-size*. GD is stopped when a stationary point (i.e., $\nabla \mathrm{F}(\mathbf{x}^*) = \mathbf{0}$) is reached, since there is no improving direction to follow. GD-based methods are capable of converging to the global optimum solution only when the initial solution is properly chosen, that is, it is very close to the global optimum. Due to their strictly deterministic nature, GD belongs to the class of local optimization methodology. As a matter of fact, GD-based methods converge to the global optimum when the initial solution is very close to the global optimum itself; otherwise, there is no way that these methods converge to the desired solution. Figure 2.2 depicts the Ackley function [4], which is an example of multi-modal objective function, with the global minimum in $\mathbf{0}$ and several local minima. Notice that the Ackley function was extended to arbitrary dimensions in [25] with the following equation:

$$\mathrm{F}(\mathbf{x}) = -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)) + 20 + e, \quad -32 \leq x_i \leq 32.$$

For this function, GD converges to **0** only when the initial solution is very close **0**. In order to overcome the aforementioned limitation, some modification have been proposed, such as the Multi-Start strategy. In such a case, a set of initial solutions is generated and they are used as starting points for the optimization procedure. Since a starting point could be sampled close to the global solution, this strategy increases the probability of converging to the global optimum. Nevertheless, the performance of the Multi-Start strategy is not good when large, noisy or high dimensional search spaces are taken into account.

2.1.3 Other classic optimization techniques

A plethora of different optimization techniques has been presented in literature, such as Random Optimization [279], Simulated Annealing [228] and Tabu Search [161, 162].

Random Optimization

Random Optimization (RO), known also as direct-search or derivative-free, is an ensemble of local numerical optimization techniques that do not require continuous and differentiable functions, since they are not based on the gradient. RO is an iterative procedure that moves a random solution $\mathbf{x}^*$ to better positions in the search space by applying some statistical distributions (e.g., normal distribution). The basic scheme of RO can be summarized as follows:

1. initialize the candidate solution $\mathbf{x}^*$ with random position in the search space;
2. calculate a new position $\mathbf{x}' = \mathbf{x} + \mathbf{r}$, where $\mathbf{r}$ is a vector of random numbers sampled from a normal distribution;
3. if $\mathrm{F}(\mathbf{x}') < \mathrm{F}(\mathbf{x}^*)$, then set $\mathbf{x}^* = \mathbf{x}'$; otherwise discard $\mathbf{x}'$;
4. if the termination criterion is met, then stop the RO and return $\mathbf{x}^*$; otherwise go back to *Step* 2.

At the end of the procedure, $\mathbf{x}^*$ is the best solution. Matyas [279] proved that the basic version of RO is capable of converging to the global optimum of simple unimodal functions when an infinite number of iterations are performed, a condition that makes RO unfeasible in real scenarios. Baba [23] and Solis [405] showed that different variant of RO, exploiting other statistical distributions, are able to convergence to the region of the global optimum.

Simulated Annealing

Simulated Annealing (SA) is a global optimization method belonging to the probabilistic metaheuristics; as in the case of RO techniques, to explore the search space SA does not require that the function is continuous and differentiable. It takes inspiration from the annealing in metallurgy, that is, the melting of metal and its successive slow cooling. While the material cools down, the state of the system $\mathbf{x}^*$, which corresponds to a candidate solution of the given optimization problem, can probabilistically either change to a neighbor state $\mathbf{x}'$ or remain in the same state. This procedure is iterated until the lowest energy configuration is met. The neighborhood of $\mathbf{x}^*$ is strictly dependent on the structure of the solutions. A common way to define the neighborhood is to build a set of random points contained in a hypersphere centered in $\mathbf{x}^*$. All the solutions $\mathbf{x}^*$ in the search space are characterized by a different energy value (i.e., the value of the objective function) that is denoted by $E(\mathbf{x}^*)$. In a probabilistic way, the system moves towards states (i.e., solutions) with lower energy, but also new solutions with higher energy can be accepted. The acceptance of those solutions is calculated as follows:

$$P(\text{accept } \mathbf{x}' | E(\mathbf{x}^*), E(\mathbf{x}')) = \begin{cases} 1, & \text{if } E(\mathbf{x}^*) > E(\mathbf{x}') \\ \exp \frac{-|E(\mathbf{x}^*) - E(\mathbf{x}')|}{T}, & \text{otherwise} \end{cases}, \tag{2.4}$$

where T is the current temperature of the system, which generally starts from 1 decreasing to values very close to 0 during the last iterations of the algorithm. Thanks to this simple strategy, SA has a less chance of being trapped in local minima. In [228] the authors proved that SA asymptotically converge to the global optimum. SA can be seen as an important improvement with respect to traditional local search techniques thanks to the exploration phase of the search space that is used instead of a promising neighborhood.

Notice that SA is an adaptation of the Metropolis-Hastings algorithm [289], which is applied to generate sequences of random number from multivariate distributions.

Tabu Search

Tabu Search (TS) is an extension of SA, so it is a metaheuristics exploiting a neighborhood search procedure that iteratively proceeds from a solution $\mathbf{x}^*$ to another solution $\mathbf{x}'$, which belongs to the neighborhood set $\mathcal{N}(\mathbf{x}^*)$ of $\mathbf{x}^*$, until a termination criterion is met. In order to explore as much as possible the regions of the search space, each solution $\mathbf{x}' \in \mathcal{N}(\mathbf{x}^*)$ is reached from $\mathbf{x}^*$. TS exploits the following memory structures to carry out the optimization:

- *short-term.* It is the list of the solutions that have been recently considered. These solutions cannot be revisited until they are in the list;
- *intermediate-term.* It contains some intensification rules that are used to bias the search towards the promising areas of the search space;
- *long-term.* It is the structure in which are stored the diversification rules driving the search procedure into new regions.

These structures allow for iteratively moving from the current solution $\mathbf{x}^*$ to an improved solution $\mathbf{x}' \in \mathcal{N}(\mathbf{x}^*)$. The ensemble of these memory structures forms the so-called tabu list containing both rules and “banned” solutions that are exploited to add novel solutions to the neighborhood $\mathcal{N}(\mathbf{x}^*)$ of $\mathbf{x}^*$.

Short-term memory stores the selected attributes (called tabu-active) of the recently visited solutions. All the solutions that have tabu-active attributes are “banned". In order to override the tabu state of a solution, some aspiration criteria can be adopted. The most common used aspiration criterion allows for taking into consideration those solutions that are better than the current best solution, according to the objective function. The most simple rule that can be added to the *intermediate-term* memory allows for preventing those solutions containing certain attributes (e.g., undesirable values of certain variables). Finally, in the *long-term* memory are stored some functioning settings, such as the total number of iterations since the beginning of the search where the solution components have been associated to the current best solution.

Other advanced optimization techniques were designed to obtain a good balancing between the exploration and exploitation phases, which are generally controlled by means of some functioning settings. All the methodologies described in this section improve a single solution $\mathbf{x}^*$ until a convergence criterion is achieved. EC and SI techniques adopt a completely different approach in which a *population* of candidate solutions is evolved until a convergence criterion is met.

2.2 Evolutionary Computation

EC is a field of research that aims at proposing metaheuristics inspired by Darwinian evolution theory to solve complex problems [105]. All the bio-inspired EC methods that have been proposed share the following characteristics:

- the candidate solutions are represented by a population P of randomly generated individuals;
- during the generations, P evolves by means of an iterative process in which the individuals are modified by using random modifications;
- each individual in P is characterized by a fitness value—measured applying a *fitness function*—which measures its quality;
- the process is stopped when a termination criterion is met (e.g., the maximum number of generations is achieved, an optimal solution is found or the maximum execution time is reached);
- some functioning settings are used to balance the exploration and exploitation phases. A fine tuning of these settings is required to obtain the best performance.

The fitness function represents the objective function previously discussed, allowing for measuring the “goodness" of the individuals in P and driving the evolution of P. The surface (see Figures 2.2) described by the fitness function and containing the set of feasible solutions in the search space is called *fitness landscape*.

Some settings-free algorithms (such as, parameter-free Genetic Algorithms [384], Tribes [88], Proactive Particles in Swarm Optimization [309] and Fuzzy Self-Tuning Particle Swarm Optimization [305]) have been proposed to avoid the necessity of performing a fine tuning of the functioning settings.

2.2.1 Evolution Strategies

Evolution Strategies (ES) are one of first EC strategies, further developed by Rechemberg [354], proposed to solve optimization problems in continuous search spaces. In ES the number of individuals $|P|$ of the population P is traditionally denoted by μ; each individual represents a position in the $\mathbb{R}^D$ domain. In order to modify the population among the generations, ES exploits the following mechanism:

- new offspring are generated by means of recombination operators applied to a subset of individuals (called parents) of P;
- a mutation operator is applied to the generated offspring, which are then inserted in the population P;
- the population size is reduced to μ by means of a selection operator.

The number of parents that are used for the recombination phases is denoted by ρ, with $2 \leq \rho \leq \mu$, while other EC techniques exploit exactly two parents for the crossover mechanism. In each generation the number of new offspring is λ, with $\lambda \geq \mu$. Different recombination strategies can be applied to blend the parents, such as a weighted average of the ρ parents or a discrete combination of the components of the real-valued vectors encoded by the parents (the interested reader is referred to Hansen *et al.* [182] and [46], where a complete description of recombination strategies is presented).

Once the offspring have been generated, they undergo a mutation operator that is generally used to perturb the components of the real-valued vectors. The most applied perturbation is based on the multivariate normal distribution $\mathrm{N}(\mathbf{0}, \mathbf{C})$, where $\mathbf{C} \in \mathbb{R}^{D,D}$ is the covariance matrix. $\mathbf{C}$ allows for generating completely general perturbations as well as isotropic, axis-aligned perturbations. Moreover, it can adapted during the iterations to take into account the correlations that can exist among the components.

Regarding the selection strategy, the truncation selection is the most common. This strategy selects the best μ individuals, i.e., those characterized by the best fitness values, for the next generation. Two specific approaches for the creation and the selection of the novel population were proposed, namely, *plus (+)* and *comma (,)*. In the former, the best μ individuals among the $\mu + \lambda$ individuals survive in the next generation, while in the latter all parents die and the best μ of λ are chosen for the next generation. Finally, a plus-selection named $(1 + \lambda)$-ES was proposed to the aim of preserving the best individual (including the parents) among the generations, which is also used to generate new offspring. There is no convergence theorem valid for every version of ES; however, there exist a convergence theorem, proven by Granville [167] for the SA approach, which is also valid for $(1, 1)$-ES method with a fixed mutation strength and a time-dependent selection pressure [45].

2.2.2 Covariance Matrix Adaptation Evolution Strategy

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [184] is probably the most effective stochastic EC strategy for optimization problems in continuous search spaces. The most important feature of CMA-ES is the self-adaptation of the covariance matrix to the problem under investigation. The basic version of CMA-ES exploits a simple ES strategy that uses a mutation operator to explore the search space. The mutation operator is applied to generate the offspring by perturbing the best individual or a novel individual created by using a recombination strategy that exploits a weighted average of the population. The mutation operators consists in a perturbation obtained by using multivariate normally distributed random deviates, which have mean equal to $\mathbf{0}$ and standard deviation $\boldsymbol{\sigma}$ (named step-size). Finally, deterministic selection of the best μ individuals is applied to create the new population.

The most important modification introduced by CMA consists in dynamically adapting the multivariate normally distributed random deviates to achieve more successful mutation steps. Thus, the mutation strategy used in CMA-ES is based on a D-dimensional ellipsoid distribution, whose size and rotation are updated during the generations according to the optimization convergence. More precisely, CMA-ES relies on a $D \times D$ symmetric positive covariance matrix $\mathbf{C}$, which is adapted to capture possible existing pairwise dependencies between the components of the problem under investigation. This adaptation scheme determines the correct scaling of a given problem, allowing for invariance with respect to any orientation of the coordinate system.

CMA-ES is almost a settings-free algorithm since only the initial step-size must be provided by the user. As in the case of ES, a global convergence theorem for any version of CMA-ES is not available.

2.2.3 Differential Evolution

Differential Evolution (DE) [413] is a population-based algorithm that exploits mutation and crossover operators during the evolutionary process. Multiple versions of DE have been proposed [102, 290], generally based on the following schema:

- three different candidate solutions are randomly selected and blended by means of a weighted function (i.e., mutation);
- the obtained individual (called recombined individual) undergoes to the crossover operator along with a fourth randomly chosen individual generating a a new offspring called trial vector;
- the fitness function of the trial vector is evaluated;
- if the fitness value of the trial vector is better than that of the recombined individual, the trial vector replaces the recombined individual in the population for the next generation.

According to the classic taxonomy of DE and the general convention, the strategies are indicated as DE/*a*/*b*/*c*, where *a* is a string denoting the individual (e.g., randomly selected, the best one) that is modified by means of the mutation operator, *b* indicates the number of different individuals considered to perturb *a*, and *c* represents the type of crossover that is applied. For instance, according to [413, 142], the DE/rand/1/bin strategy exploits a differential weight $F \in [0,2]$ to balance the recombination of the three randomly selected individuals, and performs a random crossover with the parent solution by using a crossover probability $CR \in [0,1]$. To be more precise, the mutation operator is applied to a randomly selected individual $\mathbf{x}$ as follows: three individual $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ are randomly selected and are then used to calculate the recombined individual $\mathbf{z} = \mathbf{x}_1 + F \cdot (\mathbf{x}_2 - \mathbf{x}_3)$. The trial vector is obtained by applying a uniform crossover (binomial distribution) to $\mathbf{z}$ and $\mathbf{x}$, which replaces $\mathbf{x}$ in the next generation if it is characterized by an improved fitness value. Another well-known version is the DE/rand/1/exp in which an exponential crossover with self-adaptive probability is used instead of a uniform crossover.

Few studies were conducted about the conditions for the DE convergence to the global optimum. Hu *et al.* [203] proposed a sufficient condition along with a corollary for the DE convergence, showing that DE is able to converge to the global optima if the trial vectors satisfy the sufficient condition. They proposed a DE algorithm framework integrating an extra mutation component that allows for satisfying the proposed convergence condition.

2.2.4 Estimation of Distribution Algorithm

Estimation of Distribution Algorithm (EDA) is an EC method that explores the space of potential solutions by sampling and incrementally building explicit probabilistic models of the promising candidate solutions found so far [190]; therefore, the optimization can be seen as incremental updates of a probabilistic model. The basic version of EDA starts with the model that encodes the uniform distribution of the admissible solutions and ends with a refined model capable of generating global optima solutions [247]. Other versions start with a model encoding different *a priori* distributions (e.g., normal) of feasible solutions. The EDA computational framework is general-purpose and can virtually be used with any probabilistic distribution and graphical model. The rationale underlying the most basic EDA version is to iteratively generate new probabilistic models of the distribution of the parameters, which is used to iteratively sample λ new individuals. The new individuals are used to refine the distribution, according to their fitness values, and generate new offspring. In the case of Gaussian mixture models, a normal random number generator is initialized with a certain mean and standard deviation (σ) for each component. During the update phase, the distribution of the parameters is adapted and a new generation is provided. Among the existing variants of EDAs [247], a real-valued version named continuous Population-Based Incremental Learning (PBIL$_C$) [389] uses the information about the $\mu < \lambda$ best individuals found during each generation to dynamically adapt the underlying Gaussian generative model. The initial distribution is initialized on the centroid χ_0 of the search space and the standard deviation is set to σ for each component. In particular, the vector $\boldsymbol{\sigma}$ of the standard deviations is adapted by averaging the standard deviations of the μ best individuals.

A converge theorem valid for some EDA versions applied to a specific class of problems (i.e., functions that can be additively decomposed) was provided in [295].

2.2.5 Genetic Algorithms

Genetic Algorithms (GAs) represent an EC technique for global optimization tasks [125]. Inspired by the mechanisms of natural selection, GAs look for optimal solutions to complex problems by evolving a population P of randomly created candidate solutions [200]. In the most general formulation, each solution represents an individual encoded as a fixed length string of characters taken from a finite alphabet (i.e., the genes); in alternative versions, the individuals codify real-valued genes [12].

In GAs, the population P evolves in competition under controlled variation. Based on the fitness value, each individual competes according to a *selection* process, inspired by the Darwin's "survival of the fittest" laws. Given an optimization problem, a GA is initialized with a randomly generated population of individuals corresponding to the potential solutions. During each *generation* (i.e., iteration) the quality of each individual is evaluated by using a fitness function. Then, the individuals undergo to the *selection*, *crossover* and *mutation* operators to create the next population. Notice that in ES (as shown in Section 2.2.1) the selection strategy is applied after the recombination, while GAs exploit a selection procedure before the recombination.

The following schema represents a general implementation for GAs:

1. given the problem to be optimized, randomly generate an initial population;
2. for each individual, evaluate the fitness function;
3. according to the fitness values, select the individuals for the next generation;
4. apply crossover and mutation operators to recombine and vary the selected individuals;
5. if the termination criterion is satisfied, terminate the GA and return the individual with the best fitness among the population; otherwise go back to *Step* 2.

Different termination criteria can be used: for instance, the GA could stop when a given number of generations T_{max} is reached or when the fitness value of an individual is lesser than a fixed threshold value. Given the population P, consisting in $|P|$ individuals, all selection techniques have the same goal, that is, obtaining an intermediate population P' by means of a fitness-dependent sampling procedure [24], composed of copies of individuals from the previous population P. Each individual can be added more than one time at P', depending on its fitness value (the better the fitness, the higher the chance to be selected).

Once P' has been created by means of any *selection* procedure, the variation operators (i.e., crossover and mutation) are applied to obtain new individuals, named offspring, for the next generation. Notice that the individuals characterized by a better fitness will be selected with higher probability, by using one of the available selection strategies: *roulette wheel selection* [163], *ranking selection* [26] and *tournament selection* [293].

The *crossover* operator is used to recombine the genetic information from two parent individuals (Parent_1 and Parent_2). This operator, which plays a key role in GAs, allows for obtaining offspring solutions with better characteristics with respect to the parents. Crossover is applied to several selected pairs of individuals with a probability value p_c, named crossover rate. There are different versions of crossover, such as *single point crossover*, *two point crossover* and *uniform crossover*. In order to arbitrarily alter one or more genes of a selected offspring the *mutation* operator can be applied, increasing the variability inside P'. This simple operator, which allows for introducing new genetic material into the population, is able to prevent the convergence of the individuals to local optimal solutions, increasing the probability to reach every point in the search space. Also this operator is applied with a probability p_m (i.e., mutation rate), such that some randomly selected genes of each individual are selected and their values are perturbed. Generally, the crossover is executed with high probability values (e.g., $p_c = 0.95$), conversely the mutation is applied with very low rates (e.g., $p_m = 0.05$). These parameter settings are chosen so as to reflect the natural evolution process in which the characteristics (genes) of two parents are deeply blended, while mutations has a very low probability to occur. Taking advantage of the information accumulation about the unknown search space underlying the fitness function, relying on a sufficient large population, GAs are capable of converging to optimal solutions, by adapting the individuals during the iterations through exploration and exploitation of the promising subspaces. The *scheme theorem* proves that GAs asymptotically converge to the optimal solution of any problem under investigation [200].

Finally, the *elitism* strategy could also be exploited. This strategy is applied to copy the best individual (or a subset of best individuals) directly into the next generation without modifying it (them). This prevents the quality of the best solution from decreasing during some iterations since the best individual is copied into the next population without undergoing the genetic operators. Notice that this strategy is similar to the plus-selection $(1+\lambda)$-ES exploited in ES (see Section 2.2.1).

2.3 Swarm Intelligence

While EC techniques are inspired by Darwinian evolution theory, SI methodologies are based on the emergent collective behavior of groups of living organisms. As a matter of fact, socio-biological investigations showed that some animals and social insects (e.g., ants, bees, bats) [201] are characterized by some behavioral patterns that allow these animals to share information, self-organize and perform complex tasks that cannot be conducted by single individuals [390]. These peculiarities have been exploited to design and propose several nature-inspired optimization techniques, such as those described in the next sections, namely: Ant Colony Optimization, Artificial Bee Colony, Bat Algorithm, Particle Swarm Optimization and its settings-free versions [309, 429, 305], called Proactive Particles in Swarm Optimization and Fuzzy Self-Tuning Particle Swarm Optimization.

A convergence theorem does not exist for these approaches, expect for a class of Ant Colony Optimization algorithms [416].

2.3.1 Ant Colony Optimization

Ant Colony Optimization (ACO) [114] is probably one of the most used SI approaches to solve combinatorial problems, such as the Traveling Salesman Problem (TSP) [415] and the Maximum Independent Set [260]. As a matter of fact, in [175] the authors proposed convergence theorems for ACO, proving its efficiency and effectiveness in optimizing solutions for NP complete problems. ACO is based on the stigmergy mechanism, which is indirectly used by the pheromone-based ants as signaling to communicate [436]. The most-known pheromone-based ants are the foraging ants, which deposit pheromone trails along the routes leading to a food source. Once an ant meets one of these pheromone trails, it tends to travel along that route aiming at reaching the food, reinforcing the signal of that route. In such a way, only the optimal route automatically emerges from the collective movement of the colony. ACO method emulates the behavior of the real ants by using simulated pheromone trails that are stochastically generated to iteratively improve the set of candidate solutions.

2.3.2 Artificial Bee Colony

Artificial Bee Colony (ABC) is another population-based SI optimization algorithms, which exploits the emergent behavior of foraging honey bees [223]. In this optimization method, three different groups of honey bees (i.e., scouts, onlookers, and employees) compose the colony and cooperate in identifying the best food resources, which correspond to the solutions with the best fitness values. Employees (also called workers) and onlookers carry out the exploitation process, while scouts lead the exploration process. To be more precise, scouts are randomly distributed across the search space and become employees as soon as they identify a promising food source, which corresponds to a region of the search space characterized by solutions with good fitness values. Onlookers are divided into groups and assigned to the employees, proportionally to their fitness values. During the iterative process, they randomly search around the food sources previously found by the employees, looking for more promising solutions. When onlookers cannot improve their position anymore, the food source is abandoned and bees return to the hive to start a new search for food by randomly choosing a new position. ABC is a global optimization method that was generally applied to real-valued problems, showing to be competitive with respect to other SI and EC techniques [222].

2.3.3 Bat Algorithm

The Bat Algorithm (BA), originally presented in [481], is a metaheuristic based on the echolocation behavior of microbats. Microbats are able to emit very loud and short sound pulses and analyze the echo that the surrounding objects reflect back. This information is used to discriminate the direction of their flight to avoid collisions. The optimization algorithm created by using this biological metaphor can be seen as a combination of an exploration procedure —where the movement of bats is influenced by a frequency value—and a local search controlled by the loudness and rate of pulse emissions. Microbats adjust both the frequency of their emitted pulses and the rate of pulse emission, according to the proximity of the preys. Moreover, also the loudness fluctuates along with the proximity of preys, that is, when a bat is near its target its loudness decreases. Each bat in the swarm can find the best positions by performing an individual search or moving to the best location previously found by the swarm.

To be more precise, the original version of BA is based on the following criteria: (*i*) the echolocation is used by bats to sense distances and to discriminate between prey and background barriers; (*ii*) each bat i is characterized by a position $\mathbf{x}_i$, flies with a velocity $\mathbf{v}_i$ (equal to 0 at the first iteration), and emits on a frequency f_i varying in the range $[f_{min}, f_{max}]$. In addition, each bat can vary its own loudness A_i and pulse rate r_i when its fitness is improving. The loudness A_i varies within the interval $[A_{min}, A_0]$, where A_0 is a very large positive value, while the pulse rate r_i can take values in the range $[0, 1]$.

At each iteration t of BA, each bat updates its frequency, velocity and position (with respect to the previous iteration) as follows:

$$\begin{aligned} f_i^t &= f_{min} + (f_{max} - f_{min})\beta, \\ \mathbf{v}_i^t &= \mathbf{v}_i^{t-1} + (\mathbf{x}_i^{t-1} - \mathbf{x}^*) f_i^t, \\ \mathbf{x}_i^t &= \mathbf{x}_i^{t-1} + \mathbf{x}_i^t, \end{aligned} \tag{2.5}$$

where f_{min} and f_{max} depend on the size of the search space, β is a random number drawn from a uniform distribution in $[0,1]$, and $\mathbf{x}^*$ is the current global best solution.

For what concerns the local search phase of BA, each bat generates a new solution around the best solution found during the current iteration $\mathbf{x}_{best}^t$ by using the random walk $\mathbf{x}_{new} = \mathbf{x}_{best}^t + \varepsilon A^t$, where $A^t = \langle A_i^t \rangle$ is the average loudness of all bats at iteration t, and ε is a random number drawn with a uniform distribution in the range $[-1,1]$. Finally, at each iteration and for each bat i, the loudness A_i^t decreases as $A_i^t = \alpha A_i^{t-1}$ if $rnd < A_i^t$, where α is a fixed parameter and rnd is a random number sampled with uniform distribution in $[0,1]$. Similarly, the pulse rate r_i^t increases as $r_i^t = r_i^0(1 - e^{(-\gamma t)})$, where γ is a fixed parameter and r_i^0 is the initial pulse rate. This increment is performed only if $\mathrm{F}(\mathbf{x}_i) < \mathrm{F}(\mathbf{x}^*)$.

An improved version of BA was proposed by Xie *et al.* [474], in which the algorithm makes use of a differential operator and Lévy flights trajectories (DLBA). The differential operator introduced in DLBA modifies the original frequency update strategy employed by bats. Lévy flights are Markov processes that differ from regular Brownian motion and allow for improving the convergence speed and accuracy of BA. Besides the position update of the bats, in DLBA also the local search process is different from the original algorithm; before the position update, each bat i performs the Lévy flight: $\mathbf{x}_i^t = \hat{\mathbf{x}}_i^t + \mu[rand - 0.5] \odot Levy$, where $\hat{\mathbf{x}}_i^t$ is the best solution found so far by bat i; μ and $rand$ are random numbers drawn from a uniform distribution in $[0,1]$; $\odot$ is the entry-wise multiplication (Hadamard product); $Levy$ is the step length that obeys Lévy distribution $Levy \sim u = t^{-\lambda}$ (with $1 < \lambda \leq 3$). The updating formulas of the frequencies and position of each bat are inspired by the mutation operator of the DE algorithm [349]. The frequencies of bats change, at each iteration, as follows:

$$\begin{aligned} f_{1i}^t &= \left((f_{1,min} - f_{1,max}) \frac{t}{n_t} + f_{1,max} \right) \beta_1, \\ f_{2i}^t &= \left((f_{2,max} - f_{2,min}) \frac{t}{n_t} + f_{2,min} \right) \beta_2, \end{aligned} \tag{2.6}$$

where β_1, β_2 are random numbers drawn from a uniform distribution $[0,1]$. $f_{1,min}$, $f_{2,min}$, $f_{1,max}$, $f_{2,max}$ and n_t are user-defined values.

The position $\mathbf{x}_i^t$ of each bat is updated as $\mathbf{x}_i^t = \mathbf{x}_{best}^t + f_{1i}^t(\mathbf{x}_{r1}^t - \mathbf{x}_{r2}^t) + f_{2i}^t(\mathbf{x}_{r3}^t - \mathbf{x}_{r4}^t)$, where $\mathbf{x}_{best}^t$ is the current best global solution and $\mathbf{x}_{rj}^t$ (with $j = 1, \dots, 4$) are the positions of four random bats in the swarm. In the final phase of the iteration, the best bat in the swarm is selected and, if the fitness value is better than $\mathbf{x}^*$, then its loudness A_i^t is decreased as $A_i^t = \alpha A_i^{t-1}$ and the pulse rate r_i^t is increased as $r_i^t = r_i^{t-1}\left(\frac{t}{n_t}\right)^3$.

2.3.4 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a stochastic population-based metaheuristic belonging to SI methods and suitable for real-valued optimization problems [225, 123]. A population (called swarm) of n widely candidate solutions (called particles) moves through a D-dimensional bounded search space and cooperates to identify the optimal solution, eventually clustering in regions where minima are identified. Each particle i, where $i = 1, \dots, n$, is identified by a position vector $\mathbf{x}_i$ of real-valued coordinates, defined in a multi-dimensional search space $\mathbb{R}^D$, and by a velocity vector $\mathbf{v}_i \in \mathbb{R}^D$ used to update its own current position. During the PSO execution, particles are attracted towards the best position $\mathbf{b}_i \in \mathbb{R}^D$ found by the particle itself so far, and the global best position $\mathbf{g} \in \mathbb{R}^D$ found by the neighborhood of the particle or by the entire swarm. The optimization process proceeds until a termination criterion is met (such as a maximum number of iterations is achieved). The social factor $c_{soc} \in \mathbb{R}^+$ and the cognitive factor $c_{cog} \in \mathbb{R}^+$ are used to balance the exploration and exploitation strategies, respectively. Moreover, two vectors $\mathbf{r}_1, \mathbf{r}_2 \in \mathbb{R}^D$ of random numbers, sampled with uniform probability in $[0, 1)$, are exploited to add stochasticity to particles movement and thus preventing premature convergence to local optima. To avoid chaotic behaviors in the swarm, the velocity of each particle is weighted by an inertia factor $w \in \mathbb{R}^+$ [187], and limited by maximum velocity values $\mathbf{v}^{max} \in \mathbb{R}^D$ proportional to the distance between the boundaries of the search space. A vector of minimum velocities $\mathbf{v}^{min} \in \mathbb{R}^D$ is often used to prevent stagnation and to keep a high diversity inside the swarm [2]. Considering what has been stated above, the velocity of each particle i is updated as follows:

$$\mathbf{v}_i = w \cdot \mathbf{v}_i + c_{soc} \cdot \mathbf{r}_1 \odot (\mathbf{x}_i - \mathbf{g}) + c_{cog} \cdot \mathbf{r}_2 \odot (\mathbf{x}_i - \mathbf{b}_i),$$

where $\odot$ denotes the component-wise multiplication operator. Afterwards, the particle position $\mathbf{x}_i$ is updated as:

$$\mathbf{x}_i = \mathbf{x}_i + \mathbf{v}_i.$$

The inertia value w can be either constant throughout the optimization or decreased by applying an update function. Since these functioning settings have a relevant impact on the optimization performance, they were deeply analyzed in several works (see [109, 21, 62] for c_{soc} and c_{cog} analyses, and [77] for w). Nevertheless, these functioning settings are problem dependent since every fitness landscape has different characteristics. Moreover, in order to avoid divergence of candidate solutions towards infinity, the search space is generally bounded. β_d^{min} and β_d^{max} (with $d = 1, \dots, D$) indicate the boundaries along the d-th dimension of the search space. Once a particle moves outside the search space, it is moved back to the valid regions of search space. This strategy allows for always evaluating the fitness function since the particles always identify valid solutions to the optimization problem. The most applied boundary conditions are: absorbing, reflecting and damping strategies [475]. Absorbing condition consists in relocating at the boundary of the search space those components of the particle positions that moves outside the valid search space, setting to zero the relative velocity components. Similarly, in the reflection condition the velocity components are reversed instead of being set to zero, while in the damping condition the velocity is reversed and modulated by using a random number sampled with the uniform distribution in the range $[0,1]$.

Even though a convergence theorem was not demonstrated, PSO was applied to solve a lot of real-world problems thanks to its implementation simplicity and optimization performance [344].

2.3.5 Proactive Particles in Swarm Optimization

As described in the previous section, PSO relies on specific functioning settings (i.e., c_{cog}, c_{soc}, w, $\mathbf{v}^{min}$, $\mathbf{v}^{max}$), which are strongly problem dependent. Determining the best values of these functioning settings is a difficult task, often unfeasible, since a complete knowledge of the fitness landscape would be required [275]. To overcome this limitation, Fuzzy Logic (FL) [484] can be employed during the optimization phase to dynamically adjust the functioning settings of PSO [1, 357]. The first attempt in exploiting a Fuzzy Rule-Based System (FRBS) in this context was presented by the original authors of PSO [394]. In their fuzzy version, during each iteration, the performance of the current best candidate solution and the current inertia weight are used as input of the FRBS to assess a new inertia weight for the whole swarm. Another version called Fuzzy Adaptive Turbulence in Particle Swarm Optimization (FATPSO) [2] was introduced with the explicit goal of solving the problem of premature convergence. In order to reduce the crowding of the swarm around the global best, FATPSO adaptively tunes the $\mathbf{v}^{min}$ of the particles by using FL. Tian *et al.* presented Fuzzy Particle Swarm Optimization algorithm (FPSO) [439], in which the inertia weight and the learning coefficient—a new parameter introduced to throttle the particle velocities—are dynamically adapted. In FPSO, the FRBS exploits two input variables: the deviation of fitness of the particles and the improvement of the global best. All these works show that FL is suitable for the development of self-tuning strategies for PSO, even if they only take into account a subset of the overall PSO settings. Moreover, the aforementioned methods consider w, c_{soc} and c_{cog} as *global* settings for the whole swarm. A different approach is to determine specific functioning settings for each particle during the optimization. Examples are the adaptive inertia weights provided in Gray PSO [253], Self-Regulating PSO [434] or the recently introduced Stability-based Adaptive Inertia Weight [424].

Nobile *et al.* proposed Proactive Particles in Swarm Optimization (PPSO) [309] in which the inertia, cognitive and social factors are calculated for each particle using an FRBS. In such a way, the *reactive* individuals are turned into *proactive* optimizing agents. The FRBS used by PPSO calculates the novel settings of each particle by analyzing two functions: *(i)* the distance of the particle from the global best, and *(ii)* a normalized fitness incremental factor of the particle. Thanks to the introduction of these fuzzy concepts, PPSO prevents the use of arbitrary thresholds (i.e., crisp boundaries) and naturally handles these concepts to control the behavior of the individual particles. The output variables used as consequents of the rules are: $Inertia_i$, $Social_i$ and $Cognitive_i$, clearly corresponding to the respective settings of the i-th particle in PPSO.

PPSO recalculates the values of $w_i^t, c_{soc_i}^t$ and $c_{cog_i}^t$—for each particle $i = 1, \ldots, n$ during each iteration t—by using an FRBS composed by the 9 fuzzy rules [309]. PPSO relies on the Sugeno inference method [418] to compute the crisp outputs, according to the calculated membership functions [309]. Given a set $\mathcal{R}$ of R rules, all relative to the same output variable, the Sugeno inference method calculates the final numerical value of that output variable as the weighted average of the output of all rules.

The aim of the rules that adjust the value of $Inertia_i$ is to increase the value of this variable when the particle shows a good performance (in terms of fitness), and to lower it otherwise. In alternative, $Inertia_i$ is set to a "neutral choice", when no relevant changes in the fitness value occur or the distance from the global best is small. The rules that adjust the value of $Social_i$ reduce the value of this factor either if the particle is finding better solutions for the optimization problem, or if it is near to the global best. On the contrary, when the particle is not finding better solutions or it is not close to the global best, it should rather "follow the leader", and $Social_i$ is increased. As in the case of $Inertia_i$, the FRBS assigns an intermediate value to $Social_i$ if no relevant changes in the fitness value occur. The rules defined to control the variable $Cognitive_i$ reduce the value of this factor if the distance from the global best is large. On the contrary, if the particle is not improving its fitness value or it is not far from the global best, then an intermediate value of $Cognitive_i$ can weigh the particle tendency to move in the direction of its personal best position $\mathbf{b}_i$, with respect to the effect of the social component. If the particle is finding better solutions, then the local exploration around its current position within the search space is encouraged by increasing *Cognitive*$_i$. Finally, PPSO automatically sets the maximum velocity of the particles, along each component of the search space, as:

$$v_d^{max} = \mu \cdot |\beta_d^{max} - \beta_d^{min}|, \text{ for } d = 1, \ldots, D, \tag{2.7}$$

with $\mu \in [0,1)$, with $\mu = 0.2$ as default value.

Fuzzy Self-Tuning Particle Swarm Optimization

Fuzzy Self-Tuning Particle Swarm Optimization (FST-PSO) [305] is an improved version of PPSO [309, 429] in which also the values of $\mathbf{v}^{min}$, $\mathbf{v}^{max}$ of each particle are individually modified during the iterations by means of fuzzy rules. In such a way, FST-PSO results to be a completely settings free version of PSO. In FST-PSO the $\mathbf{v}^{min}$ and $\mathbf{v}^{max}$ are automatically determined for each particle by means of two linguistic values associated to two linguistic variables. In order to clamp $\mathbf{v}^{min}$ of each particle along the d-th dimension of the search space, an additional setting $\omega < \mu \in [0,1)$ is introduced, obtaining this simple equation:

$$v_d^{min} = \omega \cdot |\beta_d^{max} - \beta_d^{min}|, \text{ for } d = 1, \ldots, D. \tag{2.8}$$

In this case, the five output variables of each particle i are: *Inertia*$_i$, *Social*$_i$, *Cognitive*$_i$, μ_i, ω_i. This new set of fuzzy rules along with the introduction of two new linguistic variables, designed to dynamically adjust the $\mathbf{v}^{min}$ and $\mathbf{v}^{max}$ of each particle, allows FST-PSO to obtained better results with respect to PPSO in terms of convergence speed and concerning the best solutions found in a set of benchmark functions [305].

Chapter 3

High-Performance Computing

3.1 Introduction

In the context of High-Performance Computing (HPC), the traditional solutions for distributed architectures are represented by computer clusters and grid computing [208]. The former exploits a set of inter-connected computers controlled by a centralized scheduler, while the latter consists in the logical organization of a set of geographically distributed and heterogeneous computing resources. In both cases, the overall computational task is partitioned into smaller sub-tasks, which are assigned to the various computing units for parallel or distributed computations. These infrastructures are particularly appealing because they usually require minimal changes to the existing source code of some given program. As a matter of fact, the computing units are generally based on classic architectures (e.g., the `x86` instruction set used in personal computers), so that the code can be easily ported, with the exception of possible modifications required for message passing. Moreover, both architectures support the *Multiple Instruction Multiple Data* (MIMD) execution paradigm, that is, all computing units are independent, asynchronous and can work on different data executing different portions of code.

Despite these advantages, computer clusters and grid computing have considerable drawbacks. On the one hand, computer clusters are expensive, require maintenance and are characterized by a relevant energy consumption. On the other hand, grid computing [139] is generally based on *volunteering*, whereby computer owners donate resources (e.g., computing power, storage) to a specific project [15, 16]. Several factors may further affect grid computing, notably the fact that remote computers might not be completely trustworthy: potentially unpublished data are transmitted to unknown remote clients for processing and the returned results might be intentionally erroneous or misleading. Moreover, no general guarantee about the availability of remote computers exists, so that some allocated tasks could never be processed.

A third way to distribute computation is the emergent field of cloud computing, in which private companies offer a pool of computation resources (e.g., computers, storage) attainable on-demand and ubiquitously over the Internet [60]. Although cloud computing mitigates some problems of classic distributed architectures—like the costs of the infrastructure and its maintenance—it is characterized by other problems, mainly the fact that data are stored on servers owned by private companies. This brings about issues of privacy, potential piracy, espionage, continuity of the service (e.g., due to some malfunctioning, DDoS attacks, or Internet connection problems), international legal conflicts, *data lock-in*, along with typical problems of Big Data. Among them, the transferring of terabyte-scale data to and from the cloud represents probably the biggest issue [19].

In the latest years, a completely different approach to HPC gained ground: the use of general-purpose many-core devices like Many Integrated Core (MIC) coprocessors [437] and Graphics Processing Units (GPUs) [319]. Noteworthy, both types of devices can be installed on common consumer computers and are characterized by a large number of computing cores (up to 72 for MICs and 3840 for GPUs). MICs are characterized by cores based on the x86 instruction set, extended with 512-bit vector instructions, inter-connected by means of a ring bus. Thanks to this architectural choice, any existing code developed for Central Processing Units (CPUs) should be easily ported to the MIC architecture.

Differently from MICs, GPUs are pervasive, relatively cheap and extremely efficient parallel many-core coprocessors, which were originally designed to accelerate the real-time rendering of computer graphics, freeing the CPU for further calculations [302]. Nowadays, even common consumer machines are equipped with GPUs, whose exceptional computing power can be exploited to obtain, with a single machine, the same performance of clusters and grid, without the need for job scheduling or the transfer of confidential information and data.

In the next sections, we briefly describe the Message Passing Interface (MPI) that we exploited to leverage the computational power offered by modern multi-core processors and computer clusters. Then, we introduce the MIC coprocessors that were used to distribute simulations performed with the Stochastic Simulation Algorithm (see Section 1.2.2). Finally, we describe in details the General-Purpose GPU (GPGPU) computing along with the Compute Unified Device Architecture (CUDA) platform that was exploited to design and implement two GPU-powered deterministic simulators (see Sections 4.2 and 4.3).

3.2 Message Passing Interface

MPI represents the *de facto* standard for message-passing communication protocol among processes on a distributed memory system. It was officially released in 1994 by a group of researchers with the aim of providing a standardization among different message-passing protocols [172, 330]. MPI is designed to work on a wide variety of parallel computing architectures (e.g., computer clusters, multi-core processors and many-core coprocessors), representing the most important model used in HPC [421]. Several open-source, well-tested and efficient implementations of MPI were proposed to encourage the design and development of portable and scalable large-scale parallel applications.

MPI consists in a communication protocol designed to leverage parallel and distributed computers by means of the programmer interface, protocol and semantic specifications that must be exploited during the implementation [171], to obtain high performance, scalability and portability. MPI defines both the syntax and semantic of different library routines to write portable message-passing programs in `C`, `C++` and `Fortran`, which are natively supported. The first version of MPI (named MPI-1) did not support shared memory at all, while the second version (MPI-2) had a distributed shared memory concept, so that the best performance were achieved over Non-Uniform Memory Access architectures (NUMA) [244], thanks to the memory locality. Starting from the third version (MPI-3), explicit shared memory programming models were introduced [150]. The MPI interface provides a virtual topology, synchronization and communications among different processes, which are mapped to cores or nodes by means of language-specific syntax. In the library, both point-to-point and send-receive operations among the processes to exchange data between process pairs, are available. Moreover, gather and reduce operations to combine partial results during the computation as well as barrier operations to synchronize the processes can be exploited. It is worth noting that point-to-point operations can be synchronous, asynchronous or buffered.

The best advantage of MPI regards its portability over different distributed memory architectures along with the achievable performance as several optimized implementations were proposed for different hardware. Moreover, MPI-2 and MPI-3 offer parallel I/O, dynamic process management and remote memory operations, while in MPI-3 non-blocking collective operations along with one-sided operations were introduced [199].

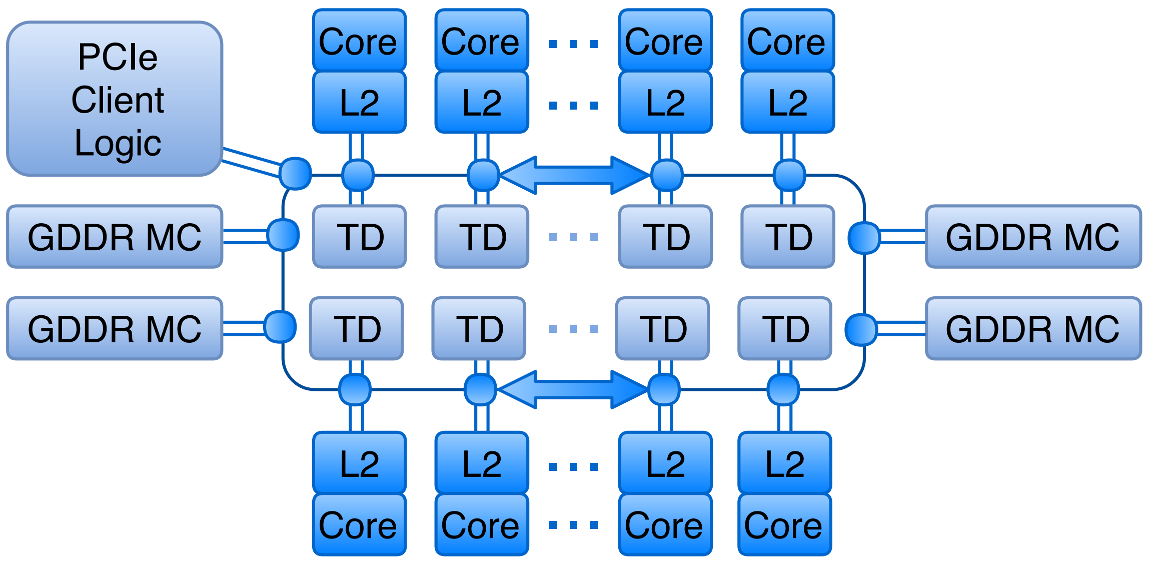


Fig. 3.1 High-level scheme of the Intel Xeon Phi coprocessor. The proposed architecture relies on a bidirectional ring that interconnects multiple processing cores (up to 72), memory controllers (GDDR MC), caches (e.g., the L2 caches that are kept coherent by TDs) and the PCI client logic interface.

3.3 Many Integrated Cores

The Xeon Phi [86] products are a family of coprocessors proposed by Intel and based on the MIC architecture. In 2012, the first family of the new line of highly parallel many-core devices, named Xeon Phi coprocessor, was launched [217]. These coprocessors were equipped with up to 61 general-purpose processor cores along with a novel powerful vector processing unit for each core.

Since every MIC consists in multiple cores based on the x86 instruction set interconnected by an on-die bidirectional ring, they can be also defined as Symmetric Multi-Processor (SMP) on-a-chip devices. As shown in Figure 3.1, this ring is also exploited to connect the Peripheral Component Interconnect (PCI) express (PCIe) interface logic and the Graphics Double Data Rate (GDDR) memory controllers (GDDR MC). Each single core of the MIC is capable of managing 4 parallel threads running in hardware and has a 512-bit wide Vector Processing Unit (VPU) with 32 vector registers per thread. Each VPU allows for a second level of parallelism by means of vector instructions that executes up to 8 double-precision operations per cycle. Moreover, in each core there is a 512-KB Level 2 (L2) cache to provide high speed and reusable data access. This cache is kept coherent by a global-distributed Tag Directory (TD) so that the cached and referenced data remain consistent across all cores. From an architectural point of view, the cores were designed to obtain a device optimized for high

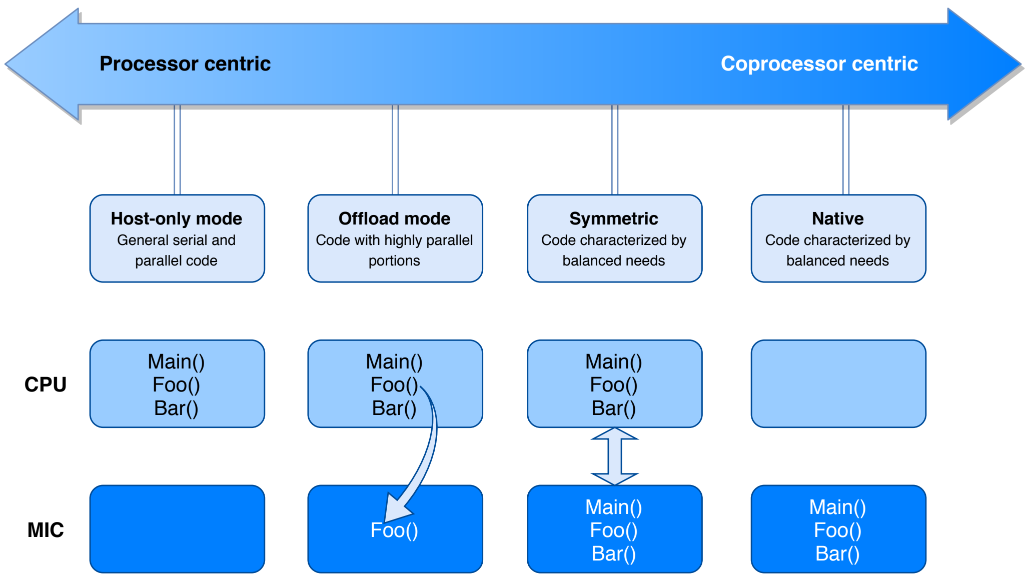

Processor centric
Coprocessor centric
Host-only mode
General serial and parallel code
Offload mode
Code with highly parallel portions
Symmetric
Code characterized by balanced needs
Native
Code characterized by balanced needs
CPU
MIC
Main()
Foo()
Bar()

Fig. 3.2 High-level scheme of the programming models that can be exploited to develop applications running on these coprocessors. In host-only mode the code is executed on the CPU (light blue boxes). When the offload mode is exploited, only the embarrassingly parallel portions of the code are run on MIC (blue boxes). In symmetric mode the code is executed on both CPU and MIC, while in native mode the entire code runs on the MIC.

levels of power efficient parallelism, while preserving the general programmability of the Intel processors. In order to reduce the size, the complexity and the power consumption they exploit a code execution based on the round-robin multithreading scheduling [352].

The Xeon Phi coprocessors exploit the PCIe system interface to comunicate with the main host computer and run over an embedded Linux μOS. Since they are equipped with an embedded system and are based on the x86 technology, they are able to leverage the existing software for Intel (and compatible) processors. Figure 3.2 depicts the main computing options to couple classic CPU (i.e., the host) and Xeon Phi coprocessors. The code can be run only on the host (host-only mode), on both processor and coprocessor (offload and symmetric modes) and only on the coprocessor (native mode). The Xeon Phi relies on two different programming models: the native mode and the the offload mode. The former is used to directly run the code on the Xeon Phi and the system bus is exploited for the communications and the data transfer between the Xeon Phi and the host. In order to develop an application running in native mode, multithreading libraries like Open Multiprocessing (openMP) and MPI are used. In the specific case of MPI, the application runs on the host and the highly parallel portions of the code are offloaded on the cores of the Xeon Phi by using compiler directives. Finally, the symmetric mode exploits cooperative communication (typically using MPI) to run the code on both coprocessors and processor cores.

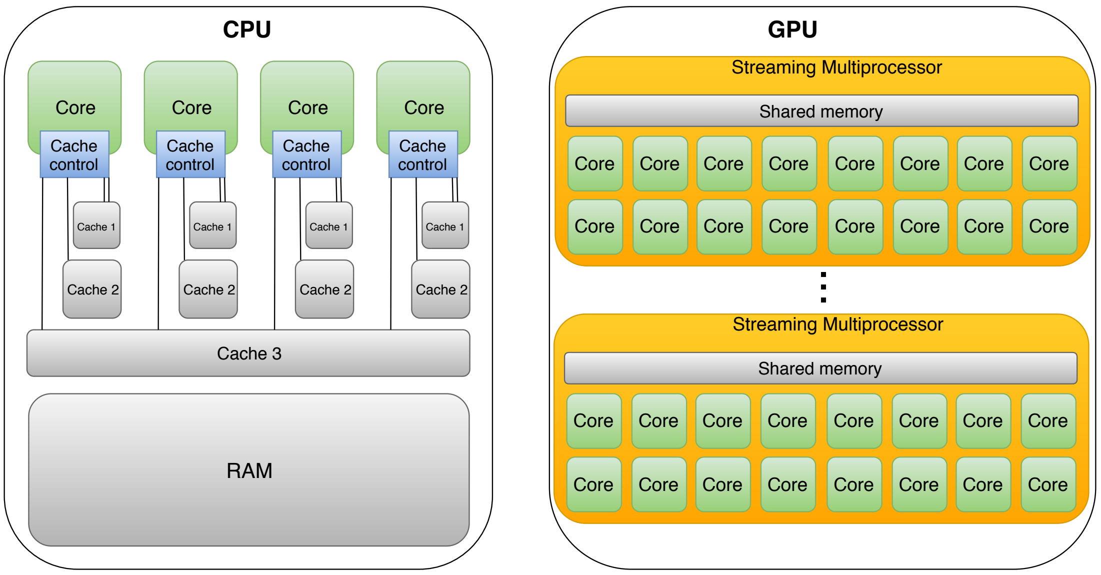


Fig. 3.3 The multi-core devices have a few numbers of cores along with different levels of cache memories, which are exploited to handle with a limited number of software threads at a time. On the contrary, many-core devices are composed of hundreds or even thousands of physical cores that are capable of managing thousands of threads simultaneously, giving access to a high level of parallelism.

3.4 General-Purpose Computing on GPUs

During the latest years, GPUs gained ground in several fields of research thanks to their ability to deal with the computational burden of intensive algorithms.

From the architectural point of view, GPUs are completely different with respect to the traditional CPUs. The former are composed of hundreds or even thousands of physical cores that are capable of managing thousands of threads simultaneously, giving access to a high level of parallelism. Conversely, the latter have a few numbers of cores along with different levels of cache memories (generally three), which are exploited to concurrently handle a limited number of software threads. Figure 3.3 provides a high-level comparison between modern CPU and GPU architectures, showing their main characteristics. From a hardware point of view, the transistors composing the CPU are used to branch predictions, caching as well as out-of-order execution optimization performed at run-time. This feature has become fundamental in the design of optimized pipelines in general-purpose processors. On the other hand, the transistors used to design the GPU architectures are mainly employed to perform strict computations since the GPUs have been produced to deal with graphic rendering pipelines at high throughput, which require a lot of independent parallel operations.

Initially, GPUs allowed for processing only a massive number of geometric primitives (especially triangles, which are ordered triples of vertices) with several attributes by using a multithread design and pipelines. They were composed of a fixed functionality set used to vertex transform and lighting (T&L) by means of few libraries (e.g., OpenGL and Microsoft Direct3D). Thanks to the increasing requests of video-game developers and 3D artists, the GPU hardware was modified by adding programmable vertex and fragment processors. Pixar in 1988 [207] introduced the naming convention for GPUs, so the programs for custom T&L were called shaders. In 2001 Nvidia proposed the first programmable GPU, motivating the creation of new languages that allowed for simplifying the development of shaders. By so doing, in 2003 Nvidia proposed its own shading language named `CG`, which was inspired by the `C` language [276]. During the same year, Microsoft added the High-Level Shading Language (HLSL) to the DirectX libraries [168]. A year later, the Khronos Group consortium proposed the shading language GLSL (OpenGL Shading Language), which was an extension of the specification of OpenGL Application Programming Interface (API). In the following years, the purpose of shaders changed from the graphics-oriented geometric transformations to general computation, giving birth to the modern GPGPU. The first general shaders were difficult to design and develop since both the input and output of the computation had to be graphics objects (such as textures, pixels or 3D meshes), while the required computations had to be performed by means of the graphics API. The following APIs introduced for general-purpose computations solved this problem, making the GPGPU computing the main alternative to the traditional HPC infrastructures.

The emerging field of GPGPU is nowadays a methodology combining CPUs and GPUs to address the increasing demand of parallel and throughput intensive algorithms. Accordingly, GPGPU takes advantage of the great computational power of modern many-core GPUs. Generally, the serial portions of the algorithms are performed on the CPUs since they are more effective and efficient than GPUs for serial processing, while GPUs perform those portions requiring the elaboration of large blocks of data that can be processed in a parallel fashion. This HPC methodology gives access to low-cost and energy-efficient solutions that allow for achieving tera-scale performances on common workstations (and peta-scale performances on GPU-equipped supercomputers [220, 50]), obtained by leveraging the powerful parallel capabilities of modern video cards.

Despite the relevant performance-per-watt and performance-per-price ratio, also GPGPU computing has some drawbacks. GPUs are mainly designed to provide the *Same Instruction Multiple Data* (SIMD) parallelism. In such a case, differently from the MIMD architecture, all cores in the GPU are supposed to execute the same instructions on different input data: this is not the usual execution strategy for existing CPU implementations, therefore the

CPU code cannot be directly ported to the GPU architecture. In general, the code needs to be rewritten for GPUs, which have completely different architectures (see Figure 3.3) and support a different set of functionalities as well as different libraries. In addition, the complex hierarchy of memories and the limited amount of high performance memories available on GPUs require a redesign of the existing algorithms to better fit and fully leverage this architecture. Thus, from the point of view of the software developer, GPU programming remains a challenging task [132] due to the effort required to code the algorithms.

Nevertheless, GPGPU computing represents a valuable and even “green” alternative to traditional HPC infrastructures: indeed, GPUs offer the possibility of creating supercomputers that are orders of magnitude faster than conventional clusters [220, 50], but having a comparable energy consumption. Empirical comparisons of equivalent implementations have also shown that GPUs attain better performances than MICs [72, 395, 427], especially when the code is not purposely modified to leverage the additional MICs throughput allowed by vector instructions. For these reasons, although GPUs require a relevant programming effort, they still represent one of the most attractive alternatives for Life Science-oriented HPC.

3.4.1 Compute Unified Device Architecture

Among the existing libraries for GPGPU computing, the most exploited are CUDA (created by Nvidia), OpenCL (proposed by the Khronos Group) and Microsoft DirectCompute.

Here we focus on CUDA, a parallel computing platform and programming model, programmable using the `C` language and based on many-core *Streaming Multiprocessors* (SMs). The CUDA compiler `nvcc` subdivides this “extended” `C` code into two separate parts: the host (i.e., the CPU) code and the device (i.e., the GPU) code. The former is compiled using any `C++` compiler that is available on the machine, while the latter is processed by using the proprietary Nvidia compiler. The device code is then embedded as binary image into the host object file. Finally, during the linking stage, CUDA runtime libraries are added to the binary executable file.

The compilation of the device code is performed using an intermediate stage during which a special representation, namely PTX, is used. PTX can be seen as the equivalent of assembly code for a “virtual GPU”, being an instruction set architecture that is supposed to remain stable and uniform across multiple generations of GPUs. The PTX is translated for one (or more) specific target architectures, supporting the capabilities of this “virtual GPU’. A schematization of this process is reported in Figure 3.4. It is worth noting that this process can be performed automatically by `nvcc` or split into two separate phases. In the second case, the intermediate PTX files can be modified before the compilation in order to drive a desired

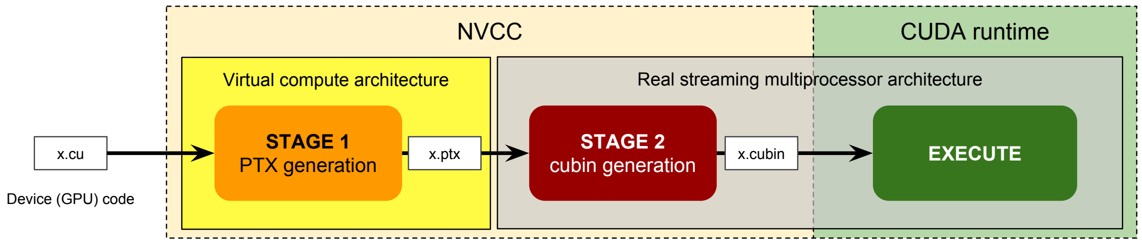


Fig. 3.4 Two-stage compilation with virtual and real architectures. Figure adapted from [312].

Table 3.1 Main libraries available for CUDA.

Name of the library	**Purpose of the library**	**Reference**
cuBLAS	Linear algebra	[314]
cuFFT	Fast Fourier transform computation	[316]
cuRAND	Random numbers generation	[317]
cuSPARSE	Linear algebra subroutines for sparse matrices	[318]
NPP	Primitives for image and signal processing	[320]
NVBIO	High-throughput sequence analysis	[313]
Thrust	Library of common data structures and parallel algorithms (e.g., sort, scan, transform and reduction operations)	[198]

behavior or to leverage special device instructions, which might not be directly supported by CUDA `C` (e.g., SIMD instructions).

CUDA is nowadays a mature architecture and offers several additional libraries that make GPGPU computing easier for developers. Among them, it is worth mentioning cuBLAS [314] (which implements linear algebra routines), CURAND [317] (which provides high-quality pseudo- and quasi-random numbers generators), cuFFT [316] (a Fast Fourier Transform library) or Thrust [198] (a `C++` template library for CUDA based on the Standard Template Library). A list of CUDA libraries and their functionalities is reported in Table 3.1.

CUDA code can run on the most widespread operating systems (Microsoft Windows, Apple macOS, GNU/Linux), although it requires a Nvidia GPU. More precisely, CUDA code can be compiled but cannot be executed on host machines equipped with ATI/AMD or Intel video cards. In order to exploit GPUs made by these vendors, alternative libraries (e.g., OpenCL) must be used. Each new generation of Nvidia GPUs offers novel or improved characteristics; the subset of features supported by a specific hardware determines its *Compute*

Table 3.2 Architectural innovations and new features introduced in the CC versions of different CUDA architectures. CUDA architectures and CC are ordered from the oldest to the latest. For each architecture, the CC generally maintains or improves the functionalities provided by the previous versions.

CUDA architecture	**Compute capability**	**Architectural innovations and new features**
Tesla	1.0	Up to 512 threads per block and 65536 blocks per grid, up to 16 KB of shared memory per SM, up to 8 simultaneous blocks per SM, 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 operations
Fermi	2.0	Up to 1024 threads per block, up to 48 KB of shared memory per SM, 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
Kepler	3.0	Unified memory programming, up to $2^{31} - 1$ blocks per grid, up to 16 simultaneous blocks per SM, up to 255 registers per thread, balanced L1 cache/shared memory configuration (32+32 KB)
	3.5	Dynamic parallelism
Maxwell	5.0	Up to 64 KB of shared memory per SM, up to 32 blocks per SM
	5.2	Up to 96 KB of shared memory per SM
	5.3	Half-precision floating-point operations
Pascal	6.*X*	Base clock up to 1700 MHz with GPU Boost, 4 MB of L2 cache, 4096-bit high performance HBM2 memories
Volta	7.*X*	Atomic addition operating on 64-bit floating point values in global memory and shared memory, Tensor Core

Capability (CC). Table 3.2 reports an overview of the various CUDA architectures to date, along with their CCs and some of the novel functionalities they introduced.

CUDA can be programmed by means of `C/C++` language: following CUDA naming conventions, the developer implements a `C/C++` function (called *kernel*) that is loaded from the host (the CPU) to the devices (one or more GPUs) and replicated in many copies named *threads*. Notice that `Fortran` is natively supported by CUDA. Threads are organized in three-dimensional structures named *blocks*, contained in three-dimensional *grids* (see Figure

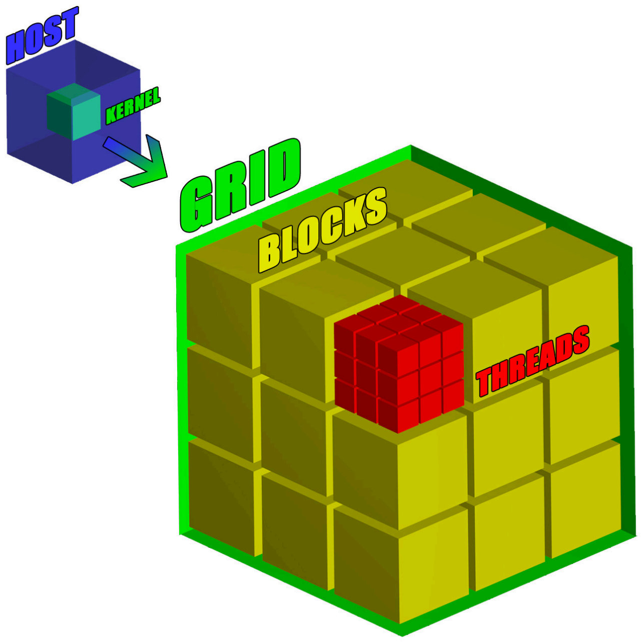


Fig. 3.5 CUDA thread organization: a kernel (represented by the light blue cube), invoked from the host (blue cube), is executed in multiple threads on the GPU (the device). Threads (red cubes) are organized in three-dimensional structures named blocks. Blocks (yellow cubes) are organized in a three-dimensional structure named grid (green cube). During the kernel launch, the programmer must specify the dimensions of both blocks and grid. Thanks to this three-dimensional arrangement, threads in a block and blocks in a grid are addressed by a triple of coordinates: the darker red thread in the figure, for instance, belongs to block $(0,0,0)$ and, inside this block, it is identified by coordinates $(1,0,0)$, being the second thread along the x axis and the first along the y and z axes.

3.5). Whenever the host computer runs a kernel, the GPU creates the corresponding grid and automatically schedules each block of threads on an available SM, allowing for a transparent scaling of performance on different devices (see Figure 3.6).

During the launch of a kernel, CUDA requires the programmer to specify how many threads T are assigned to each block and how many blocks B must be created. The total number of running threads on the GPU will be equal to $T \times B$. CUDA poses limitations to the number of threads that a single block can contain. For instance, T was limited to 512 on Tesla GPUs made by Nvidia, while it is limited to 1024 on more recent architectures (see Table 3.2). Within a block, threads can be uniformly distributed by the user along the three dimensions, given that x- and y-dimensions not exceed 1024 threads, while z-dimension must be limited to 64 threads. For instance, Figure 3.5 shows an example of three-dimensional blocks, each consisting of $3 \times 3 \times 3 = 27$ threads. The number of simultaneous threads on a SM is further limited by the available physical resources (e.g., registers, high performance

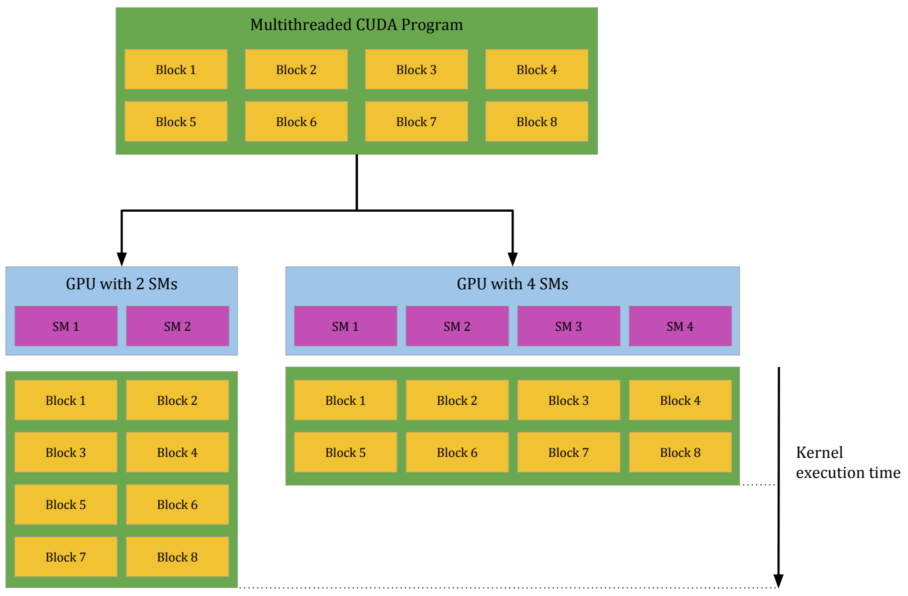
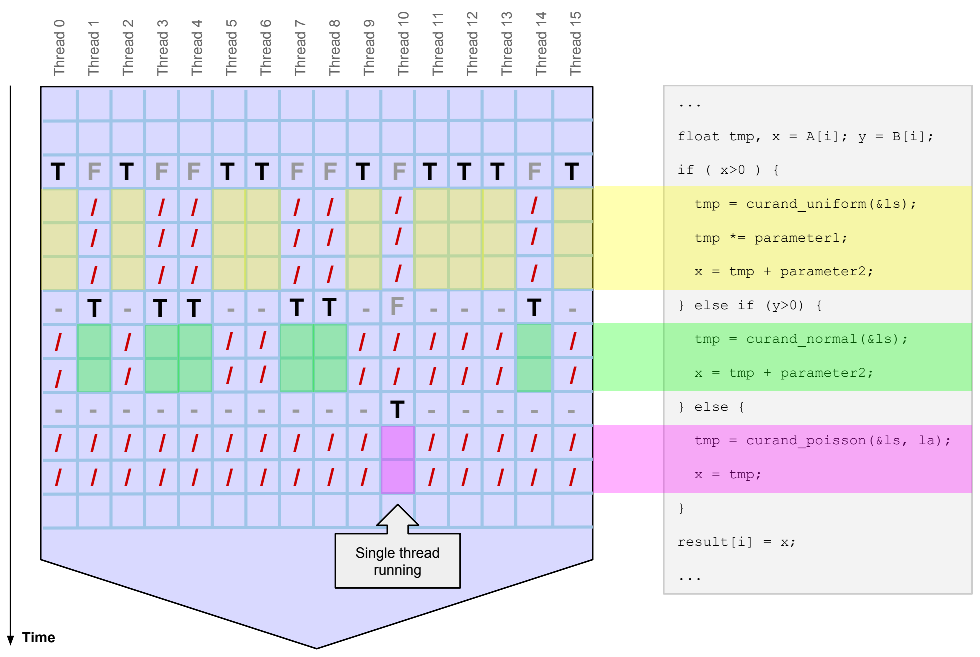


Fig. 3.6 CUDA automatic scalability: blocks are automatically scheduled on the available SMs, in a transparent way for the programmer. The more SMs are present on the GPU, the faster the execution of the kernel.

memories). It follows that the performance of CUDA kernels are tied to the choice of the values T and B.

The CUDA programming model combines SIMD and multithreading: the programmer launches a large number of identical threads that are supposed to perform identical computations on different data; however, threads are allowed for temporarily taking divergent control flows (e.g., when the code contains `if-then-else` constructs). Nvidia refers to this peculiar architecture as SIMT: *Single Instruction Multiple Threads*. Figure 3.7 provides an example of a divergent execution flow, in which 16 threads are executing a common CUDA kernel (shown in the gray box on the right). The CUDA kernel is characterized by two nested conditional branches: a first branch is taken by threads whose variable x is greater than zero (yellow region); the second branch is taken by the remaining threads, whose variable y is greater than zero (green region); the last branch is taken by the remaining threads (purple region). These three regions cannot be simultaneously executed: CUDA automatically handles the workflow by serializing the execution. Serialization affects performance, since it implies an increased running time. For this reason, conditional branches should be avoided in the kernels as much as possible. In the example shown in Figure 3.7, the last conditional branch (taken by threads that satisfy $x \leq 0$ and $y \leq 0$) causes the execution of a single thread at once



```
...
float tmp, x = A[i]; y = B[i];
if ( x>0 ) {
  tmp = curand_uniform(&ls);
  tmp *= parameter1;
  x = tmp + parameter2;
} else if (y>0) {
  tmp = curand_normal(&ls);
  x = tmp + parameter2;
} else {
  tmp = curand_poisson(&ls, la);
  x = tmp;
}
result[i] = x;
...
```

Fig. 3.7 A simple example of thread divergence. Threads are grouped according to the conditional branches, which are evaluated at run-time. Then, threads performing the same calculations are executed together (e.g., the yellow parts). In the worst case scenario, the conditional branches can cause a complete serialization of the execution (e.g., the purple parts), hence reducing parallelism and strongly affecting the overall performance.

(thread #10). It is clear that, in order to fully leverage the GPU, the algorithms might require a major redesign to reduce the recourse to conditional branches.

GPUs are equipped with a variety of memories. As schematized in Figure 3.8, the GPU memory hierarchy consists in the *global memory* (accessible from all threads), the *shared memory* (accessible from threads belonging to the same block), the *registers* (high performance on-chip memories used to store local variables), the *local memory* (part of the global memory, accessible exclusively by the owner thread), and the *constant memory* (cached and not modifiable). One additional type of CUDA memory is the *texture memory*, which is another cached and read-only memory. Formerly designed to speed up the texture mapping in real-time 3D graphics, the texture memory was later exploited by CUDA to provide cached access to non modifiable data. However, since the introduction of the Fermi architecture, the global memory is equipped with a cache system (see Table 3.2) so that the texture memory is no longer useful, except for its hardware-accelerated trilinear interpolation capabilities. Nvidia increased the L2 cache to 4 MB starting from the Pascal [322] architecture. This

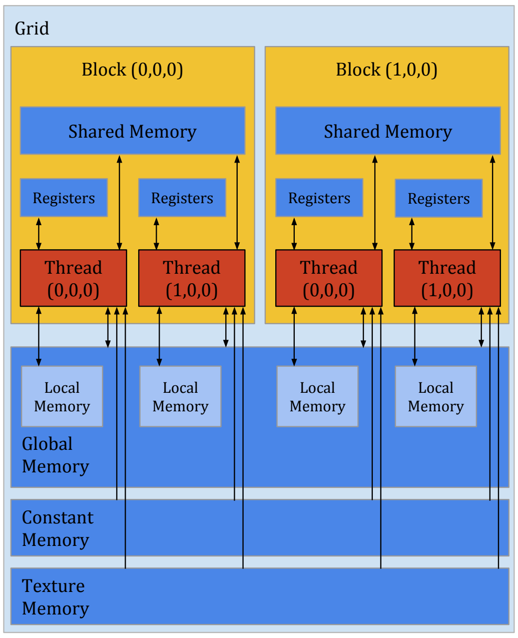


Fig. 3.8 Architecture of CUDA threads and memories hierarchy. Threads can access data from multiple kinds of memories, all with different scopes and characteristics: registers and local memories are private for each thread; shared memory let threads belonging to the same block communicate, and has low access latency; all threads can access the global memory, which suffers of 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.

increment of high performance cache reduces the number of requests to the Dynamic Random Access Memory (DRAM), improving the overall performance even on older software.

The best performance in the execution of CUDA code can be achieved by smartly distributing the data structures across these memories and, in particular, by exploiting the shared memory as much as possible. Unfortunately, the latter is a very limited resource (i.e., up to 96 KB for each multi-processor on the Maxwell GPUs, see Table 3.2), causing further restrictions on block size. On the contrary, the global memory is very large (some GB in the most recent GPUs) but suffers of high latencies, a problem mitigated by the introduction of the aforementioned caching system. Also registers may pose a limitation to performance, since they are a scarce resource. When the number of registers required by a kernel outnumbers the available physical memories, the so-called *register spilling* on global memory occurs: the local variables are stored into the (slow) global memory, affecting the overall performance. Initially, GPUs were equipped with just 63 registers per thread; with the introduction of Kepler GPUs, with compute capability 3.5, the number of threads was increased to 255 (see Table 3.2), reducing the recourse to the global memory and directly improving the performance in the case of complex kernels.

Finally, three additional matters concerning GPUs programming are worth to be mentioned. First, since blocks are asynchronously scheduled on different SMs, there exist no communication nor synchronization primitives between blocks. However, intra-block communication can be performed by means of the shared memory and the family of *warp vote* functions, which allow for the evaluation of a predicate for all threads and the recombination of the results. Intra-block synchronization is achieved using the family of `__syncthreads` functions, which can be also combined with warp voting.

Second, during the execution of a kernel, the CPU is idle and is allowed for executing additional useful tasks on the host side. This approach, which permits to fully leverage the computing power of a GPU-powered machine, is called *heterogeneous* computing. A different approach for leveraging the CPU during the execution of a kernel is the asynchronous data transfer: CUDA allows for the creation of multiple *streams*, i.e., execution queues that handle the concurrent execution of tasks, including overlapped kernel runs and memory transfers.

Third, global memory accesses can be optimized by the CUDA driver as long as the way data are organized—i.e., the access pattern—is *coalesced*, that is, the memory areas read by threads in a warp are contiguous. Although this complication has been mitigated by the most recent GPU architectures, a proper organization of data into coalesced patterns is necessary to reduce the number of memory transactions and improve the performance.

Chapter 4

High-Performance Computing for the simulation of Reaction-Based Models

In this chapter, two deterministic biochemical simulators accelerated on Graphics Processing Units (GPUs) and an implementation of the Stochastic Simulation Algorithm (SSA) on Many Integrated Core (MIC) coprocessors are described in details. The scheme in Figure 4.1 shows the “semiotic square” of the state-of-the-art of deterministic and stochastic biochemical simulators (see Section 1.2.3), accelerated by means of GPUs. This figure highlights the gap that were filled with the works proposed in this thesis—i.e., LASSIE (LArge-Scale SImulator) and FiCoS (Fine- and Coarse-grained Simulator)—that is, the development of efficient simulators capable of executing in a parallel fashion the calculations required by a single simulation and by a batch of simulations, exploiting the fine-grained or the coarse-grained parallelization strategies.

In the first section, we present a novel generator of synthetic (yet realistic) Reaction-Based Models (RBMs) of biological networks, which has been used to test the efficiency of the parallel simulators [361]. In Section 4.2, we introduce LASSIE [425], the first fine-grained deterministic simulator designed to accelerate the simulation of large-scale models (top-right panel of Figure 4.1). Section 4.3 describes FiCoS [428], an efficient deterministic simulator accelerated on GPU exploiting both the fine- and coarse-grained parallelization strategies (top-center panel of Figure 4.1). Finally, Section 4.4 is dedicated to an alternative coarse-grained implementation of SSA by means of MIC coprocessors [427] (bottom-left panel of Figure 4.1).

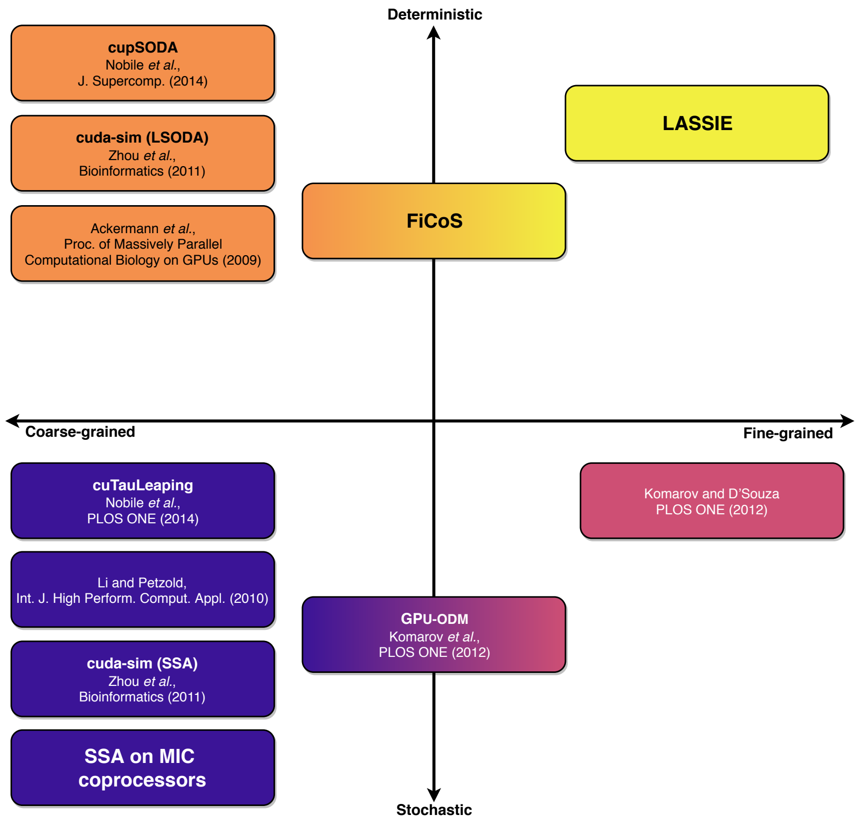


Fig. 4.1 The "semiotic square" of the state-of-the-art of accelerated biochemical simulators. The horizontal axis represents the parallelization strategy, that is, the two main approaches to leverage the threads: coarse-grained (on the left), where multiple simulations are performed in a parallel fashion; fine-grained (on the right), where a single simulation of a large-scale model is accelerated by distributing the computation over multiple threads. The vertical axis partitions the two types of simulation: deterministic (top) and stochastic (bottom). The simulators that combine either two parallelization strategies or two types of simulations are represented by means of the gradient of the colors that represent the simulators based on a single parallelization strategy or performing only a type of simulation.

4.1 SMGen

In order to test the effectiveness and efficiency of the simulators and the Parameter Estimation methodologies presented in this thesis, we designed and developed a novel custom tool capable of generating RBMs of any size. This tool, named SMGen (Synthetic Models of biological systems Generator) [361], was designed to generate synthetic (yet realistic) models of biological networks that must comply with some structural characteristics, in order for them to be as much realistic as possible:

1. system connectivity. Since a biological network can be represented as a graph, in order to guarantee a single connected component each species $S_j \in \mathcal{S}$, with $j = 1, \ldots, N$, must be involved in at least one reaction $R_i \in \mathcal{R}$, where $i = 1, \ldots, M$;

2. order guarantee. For each reaction $R_i \in \mathcal{R}$, the order of the reaction must be lower or equal than the maximum order $\max_{\text{ord}}$, whose value is a user-defined parameter. Similarly, the number of products of each reaction must be lower or equal than the user-defined value $\max_{\text{prod}}$ representing the maximum number of allowed products;

3. linear independence. In order to ensure that each reaction R_i is a plausible biochemical reaction, the reactants and products involved in R_i, represented as two vectors in $\mathbb{R}^N$, must be linearly independent;

4. unique reactions. Each reaction R_i must appear only once in the network, that is, duplicated reactions are forbidden.

SMGen saves the synthetic RBMs to file using the Systems Biology Mark-up Language (SBML) standard [229] as well as in the BioSimWare format [42], which is composed of an ensemble of input files describing all the elements necessary to define and simulate an RBM. These files have been used as input for all the simulators proposed in this thesis. SMGen is also provided with a user-friendly Graphical User Interface (GUI) that easily allows the user to provide all the settings required to generate the RBMs.

Since SMGen is designed to generate a whole set of RBMs, we developed a Master-Slave implementation to distribute and parallelize the generation of the RBMs. SMGen was entirely developed using the `Python` programming language and exploiting `mpi4py` [98], which provides bindings of the Message Passing Interface (MPI) specifications for `Python` to leverage multi-core Central Processing Units (CPUs). Figure 4.2 shows the scheme of the proposed Master-Slave implementation. The user-defined number of processes Σ are assigned as follows: Rank 0 manages the GUI; Rank 1 corresponds to the Master; Rank k, with $k = 3, \ldots, \Sigma$, are the Slave processes. Notice that SMGen requires at least 3 processes, which is the default value for Σ.

The functioning of SMGen can be summarized as follows:

1. the user interacts with the GUI (Rank 0), providing all the required settings, namely: the number N of species; the number M of reactions; the maximum order $\max_{\text{ord}}$ of the reactions; the maximum number $\max_{\text{prod}}$ of products; the probability distributions and the ranges for sampling both the initial concentrations and the kinetics parameters;

2. the GUI sends the aforementioned settings to the Master (Rank 1), which allocates the resources and offloads the network generation to the available Slaves;

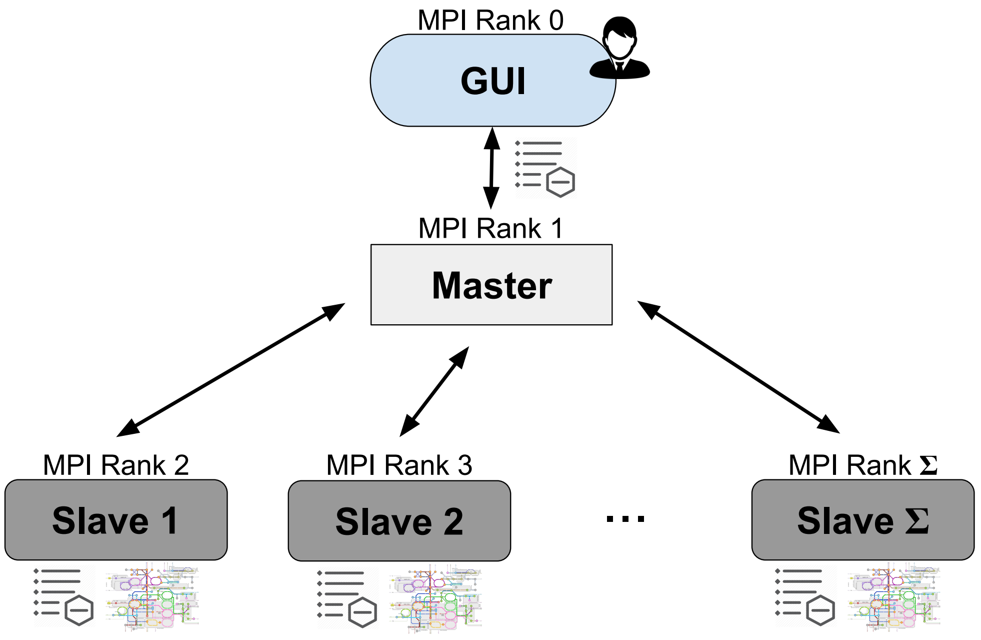


Fig. 4.2 Scheme of the Master-Slave implementation of SMGen: first, the GUI communicates to the Master the settings provided by the user; the Master orchestrates all the Slaves, sending the settings to the Slaves, which then generate the set of RBMs.

3. each Slave (Rank k, with $k = 3,\dots,\Sigma$) generates a different RBM. When a Slave finishes, it communicates to the Master that it is available and, as long as there are models to generate, the Master assigns a new network to the Slave.

A high-level overview of the algorithm underlying the generation phase, which each Slave performs, can be summarized as follows:

1. given the settings provided by the user, the graph representing the reactions is initialized;
2. the stoichiometric coefficients are randomly generated;
3. for each reaction R_i the linear independence between the reactants and products involved R_i is evaluated. If the linear independence is violated, then go back to *Step 1*. Otherwise, continue with *Step 4*;
4. for each reaction R_i any reaction equal to R_i is removed.

The algorithm is iterated until all the reactions appear only once in the network. Finally, the initial amount of the species in $\mathcal{S}$ and the kinetic constants are randomly generated by using the probability distributions chosen by the user.

4.2 LASSIE

LASSIE is the first fine-grained GPU-accelerated software designed to simulate large-scale RBMs of cellular processes, consisting in hundreds or thousands of reactions and molecular species. LASSIE was designed to be a "black-box" deterministic simulator, not requiring any expertise in mathematical modeling nor any GPU programming skill. More precisely, given the formalization of a cellular process as an RBM and assuming the mass-action kinetics [84, 78, 455] (see Section 1.1.1), LASSIE proceeds according to the following workflow:

1. it automatically generates the system of Ordinary Differential Equations (ODEs)—one ODE for each molecular species occurring in the system—according to the biochemical reactions included in the RBM;

2. it automatically derives the Jacobian matrix associated to the system of ODEs, taking advantage of the symbolic derivation, necessary to apply the Backward Differentiation Formulae (BDF) (see Section 1.2.1);

3. it executes the numerical integration of the ODEs by automatically switching between the Runge–Kutta–Fehlberg (RKF) method (described in Section 1.2.1) in the absence of stiffness and the BDF in presence of stiffness.

Notice that LASSIE was developed to solve systems of coupled ODEs specified in the form $\frac{d\mathbf{x}}{dt} = f(t, \mathbf{x})$, where $\mathbf{x} \equiv \mathbf{x}(t)$ represents the vector of concentration values at time t of all chemical species occurring in the system.

4.2.1 GPU implementation

Given an RBM as input, LASSIE automatically generates the systems of ODEs according to the Equation 1.4 and encodes the matrices $\mathbf{A}$ and $\mathbf{H} = (\mathbf{B} - \mathbf{A})^T$ as two arrays of `short4` Compute Unified Device Architecture (CUDA) vector types, named **VA** and **VH**, respectively. Notice that CUDA vector types are multi-dimensional structures ranging from 1 to 4 components, addressed by `.x`, `.y`, `.z`, and `.w`. Since the matrices $\mathbf{A}$ and $\mathbf{H}$ are sparse, LASSIE uses compressed data structures created by removing all zero elements from $\mathbf{A}$ and $\mathbf{H}$ to save memory and avoid unnecessary readings from the global memory. Namely, let h_{ji} be the element of $\mathbf{H}$ at row j and column i, and a_{ij} the element of $\mathbf{A}$ at row i and column j, for $i = 1, \dots, M$ and $j = 1, \dots, N$, where N and M are the number of species and reactions composing the RBM, respectively. For each non-zero element of $\mathbf{H}$, we store into the `.x` and `.y` components of **VH** the values j and i, respectively; the `.z` component of **VH** is used to store the element h_{ji}, while the `.w` component stores the index of the

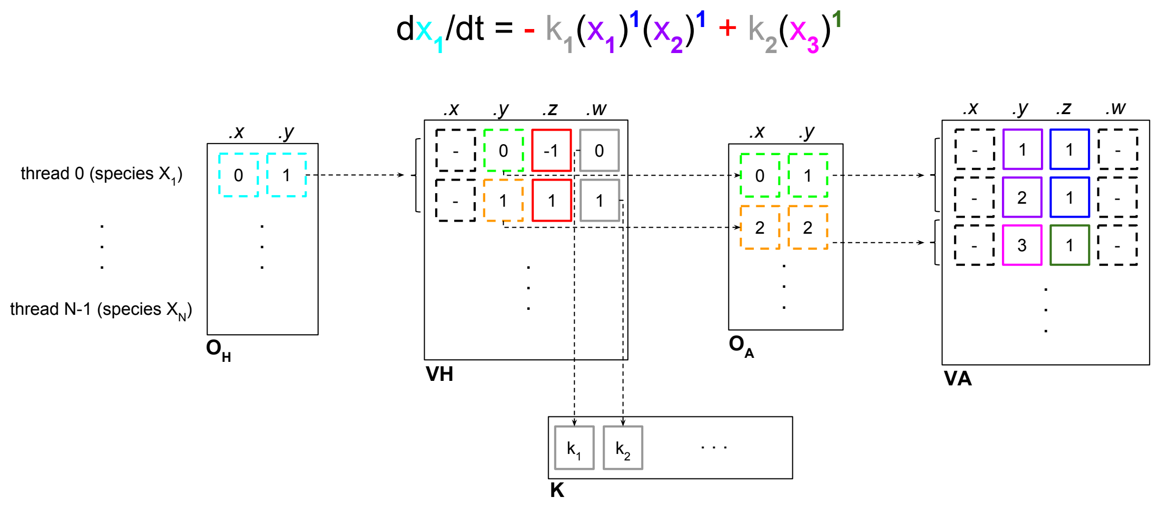


Fig. 4.3 Example of matrix encoding to automatically generate an ODE using LASSIE.

kinetic constant associated with that monomial. Similarly, for each non-zero element of **A**, the `.x` and `.y` components of **VA** contain the values i and j, respectively. The value a_{ij} is stored into the `.z` component of **VA**, while the `.w` component is left unused. Note that we exploited the `short4` CUDA vector type rather than the `short3` CUDA vector type, because the former is 8-aligned and requires a single instruction to fetch a whole entry, while the latter is 2-aligned and thus takes three memory operations to read each entry. In order to parse these arrays inside the GPU, we use two additional arrays of `short2` CUDA vector types, named $\mathbf{O_H}$ and $\mathbf{O_A}$, which store the offsets used to correctly read the entries of the **VH** and **VA** structures, respectively. The `.x` and `.y` components of each row of $\mathbf{O_H}$ contain, respectively, the first index and the last index to access the **VH** structure. Each thread uses its own pair of indeces to read the rows of the **VH** structure between the first index and the last one. Similarly, $\mathbf{O_A}$ stores the indeces that allow for correctly accessing the **VA** structure. Finally, the values of the kinetic constants are stored into an array of type `double`, named **K**. Figure 4.3 shows an example of the matrix encoding used in LASSIE, where all terms of the polynomial function describing the ODE of species x_1 given at the top of the figure are encoded in the components of the data structures $\mathbf{O_H}$, **VH**, $\mathbf{O_A}$, **VA** and **K**, as detailed hereby. Notice that only the data structures components with solid borders are used to automatically generate the ODE; the various terms appearing in the ODE are represented with corresponding colors in the data structure components. Matrix encoding starts from matrix $\mathbf{O_H}$. Each thread j, for $j = 0, \ldots, N-1$, reads the values stored in the `.x` and `.y` components of $\mathbf{O_H}$ (denoted by the lightblue borders). In this example, we consider species x_1 that corresponds to thread 0. Each thread fetches the values in **VH**, starting from the row indicated by the value stored in the `.x` component of $\mathbf{O_H}$, up to the row corresponding to the value stored in the `.y` component. In this example, thread 0 in matrix $\mathbf{O_H}$ reads the values contained in the first two rows—i.e., rows 0 and 1—in matrix **VH**. Each row of **VH** encodes a monomial of an ODE: the `.x` component is not used; the `.y` components (denoted by green and orange borders) indicate the row numbers of the $\mathbf{O_A}$ structure that each thread must read; the `.z` components (red borders) indicate the sign and the coefficient of the monomial; the $.w$ components (gray borders) indicate the positions of the array **K** containing the values of the kinetic constants corresponding to the reactions that the threads are parsing. In this example, the `.z` and `.w` components of **VH** allows for deriving the coefficients $-1k_1$ and $+1k_2$ for the first and the second term of the ODE, respectively. Afterwards, as in the case of $\mathbf{O_H}$, each thread fetches the values in **VA**, starting from the row indicated by the value stored in the `.x` component of $\mathbf{O_A}$, up to the row corresponding to the value stored in the `.y` component of $\mathbf{O_A}$. The values stored in the `.y` (violet and fuchsia borders) and `.z` (blue and dark green borders) components of **VA** correspond to the indeces of the species and the stoichiometric coefficients, respectively, while the `.x` and `.w` components of **VA** are left unused. In this example, row 0 in matrix $\mathbf{O_A}$ reads the values stored in rows 0 and 1 (`.y` and `.z` components) of matrix **VA**, generating the factors $(x_1)^1(x_2)^1$ in the first term of the ODE, while row 1 in matrix $\mathbf{O_A}$ reads the values stored in row 2 (`.y` and `.z` components) of matrix **VA**, generating the factor $(x_3)^1$ in the second term of the ODE. Therefore, in this example, the matrix encoding overall generates the ODE of species x_1 consisting in the sum of two polynomial terms: $-k_1(x_1)^1(x_2)^1 + k_2(x_3)^1$.

Thanks to these CUDA structures, we obtain a twofold performance improvement:

- at the instruction level, a single instruction is enough to either load or store a multi-word vector. By so doing, the total instruction latency for a particular memory transaction is lower and also the bytes per instruction ratio is higher;

- at the memory controller level, by using vector types a transfer request from a warp has a larger net memory throughput per transaction, yielding a higher bytes per transaction ratio. With a fewer number of transfer requests, the memory controller is able to reduce contentions producing a higher overall memory bandwidth utilization.

The only limitation due to *short* data type is that indices are limited to $2^{2\times 8}-1$, which means that LASSIE cannot simulate systems larger than 65 536 chemical species and reactions.

Once that the system of ODEs is generated and appropriately stored according to the CUDA vector types, LASSIE solves it by automatically switching between the RKF method in the absence of stiffness, and the BDF in presence of stiffness. The integration of the systems of ODEs is carried out from an initial time instant t_0, up to a given maximum simulation time t_{max}. In order to reproduce the dynamics of the cellular process described by the ODEs, the concentration values of the molecular species appearing in the RBM are saved at specified time steps within the interval $[t_0, t_{max}]$ (these time steps might correspond, e.g., to the sampling times of laboratory experiments).

The workflow of LASSIE consists in 6 distinct phases, as represented in Figure 4.4. Note that phases P_1, P_4 and P_6 are executed by the CPU (yellow boxes in Figure 4.4), while P_2, P_3 and P_5 are executed by the GPU (green boxes in Figure 4.4). Overall, phases P_2, P_3 and P_5 rely on 25 different lightweight kernels, which were specifically developed to fully leverage the parallel architecture of the GPU for the implementation of the aforementioned numerical integration methods. We describe hereafter the main design and implementation choices of each phase and their related CUDA kernels, which result in a novel parallelization strategy with respect to state-of-the-art methodologies (see, e.g., [210]).

Phase P_1. It implements the generation of all data structures used to encode the ODEs, as described above. This phase is executed on the CPU.

Phase P_2. It is used to sample and save the system dynamics and it is implemented by means of a single CUDA kernel (**kernel $\mathbf{K_1}$**). In particular, if the current simulation time t corresponds to one of the specified sampling time instants, LASSIE saves the concentration values of (possibly, a subset of) all molecular species into an array defined on the GPU. Otherwise, the execution proceeds to the next phase.

Phase P_3. It implements the RKF method [277], used by each thread j to solve the j-th ODE, for $j = 0, \dots, N-1$. This phase is implemented as 9 CUDA kernels.

During this phase, the two different approximated states $\mathbf{u}_{n+1}$ and $\mathbf{w}_{n+1}$ of the state $\mathbf{x}_{n+1}$ of the system are generated at each step, thanks to the evaluation of six supplementary values $\mathbf{l_1}, \dots, \mathbf{l_6}$ (see Table 1.4). As described in Section 1.2.1 for a single ODE, to evaluate the accuracy of $\mathbf{u}$ and $\mathbf{w}$ at the current step-size h, a user-defined vector tolerance $\boldsymbol{\varepsilon} \in \mathbb{R}^N$ (with $\varepsilon_j > 0$ for all $j = 1, \dots, N$), and two additional arrays, $\mathbf{ER}, \boldsymbol{\delta} \in \mathbb{R}^N$ are taken into account. If $ER_j \leq \varepsilon_j$ for all $j = 1, \dots, N$, then $\mathbf{u}_{n+1}$ is accepted as new state of the system, that is, $\mathbf{x}_{n+1} = \mathbf{u}_{n+1}$; otherwise, the solutions $\mathbf{u}$ and $\mathbf{w}$ are rejected and recalculated by using a new step-size. The new step-size is computed as $h = h \cdot \min\{\delta_1, \dots, \delta_N\}$, being $\delta_1, \dots, \delta_N$ the components of vector $\boldsymbol{\delta}$ (note that the new value of h has to be chosen in order to satisfy the requested error tolerance for all ODEs).

Overall, phase P_3 is implemented by means of the following kernels:

- **kernel $\mathbf{K_2}$**: used to evaluate each ODE at the current state $\mathbf{x}_n$ of the system;

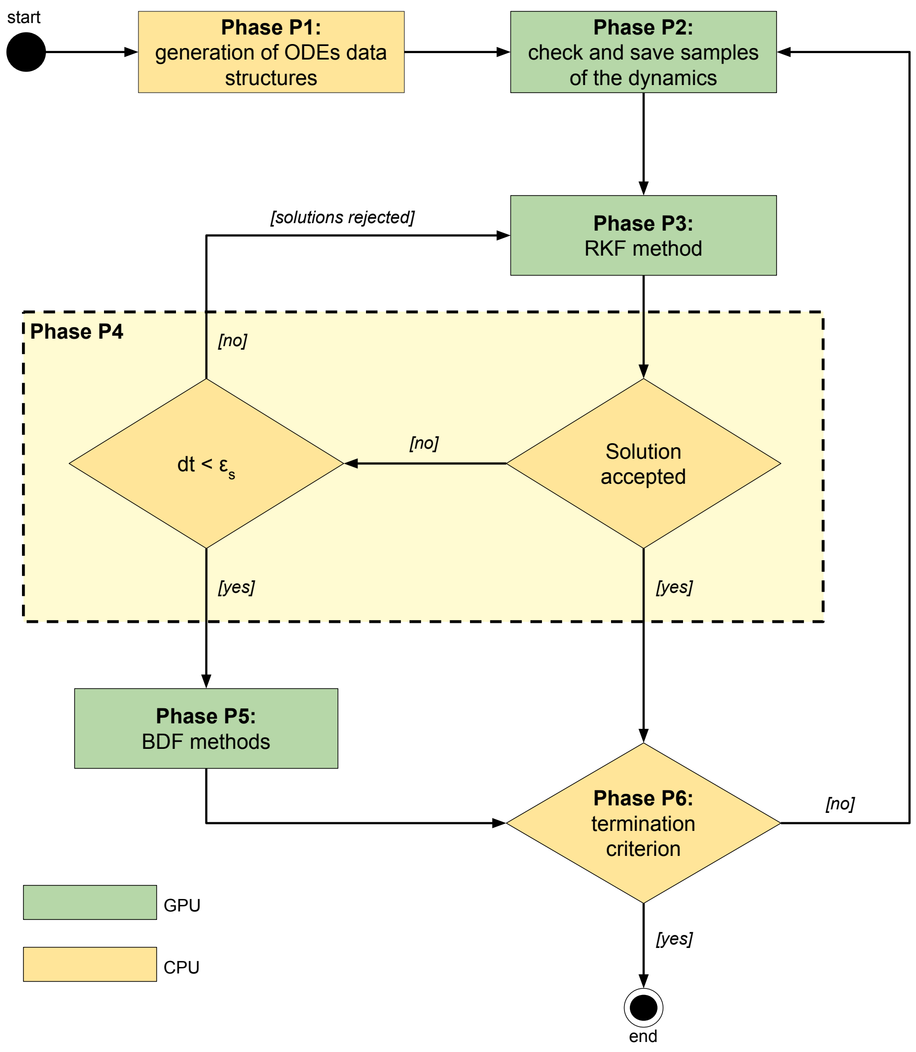


Fig. 4.4 Simplified scheme of the workflow of LASSIE. The data structures used to encode the system of ODEs are generated in phase P_1. In phase P_2, if the current simulation time t corresponds to a specified sampling time instant, then the current concentration values of all molecular species are saved; otherwise, the execution proceeds to the next phase. In phase P_3 each thread derives and solves the corresponding ODE by exploiting the RKF method, while in phase P_4 the RKF solutions are verified: (*i*) if the RKF solutions are rejected, then the integration step-size h is reduced and phase P_3 is executed again; (*ii*) if RKF solutions are rejected but the integration step-size h is too small, then phase P_5 is executed and the system of ODEs is solved by using the BDF methods; (*iii*) if the RKF solutions are accepted, the termination criterion is verified during phase P_6 (all phases from P_2 are iterated until the maximum simulation time t_{max} is reached).

- **kernels $\mathbf{K_3}$ – $\mathbf{K_8}$**: each thread j, for $j = 0, \ldots, N-1$, computes the components $l_{1j}, \ldots, l_{6j}$ of $\mathbf{l_1}, \ldots, \mathbf{l_6}$, by invoking **kernel $\mathbf{K_2}$**;

- **kernel $\mathbf{K_9}$**: each thread j, for $j = 0,\dots,N-1$, computes the components w_j and u_j of the approximated states $\mathbf{u}_{n+1}$ and $\mathbf{w}_{n+1}$, respectively;
- **kernel $\mathbf{K_{10}}$**: each thread j, for $j = 0,\dots,N-1$, calculates the components ER_j and δ_j of $\mathbf{ER}$ and $\boldsymbol{\delta}$, respectively.

Phase P_4. It is used to verify the RKF solutions calculated during phase P_3 and, accordingly, to choose the next phase to be executed:

1. if the solutions are rejected and the new step-size h is acceptable (that is, $h \geq \varepsilon_s$, for some $\varepsilon_s > 0$, e.g., $\varepsilon_s = 10^{-6}$), phase P_3 is executed again using a smaller step-size h;
2. if the solutions are rejected and the new step-size h becomes too small (that is, $h < \varepsilon_s$), LASSIE executes phase P_5;
3. if all solutions do not violate the specified RKF-tolerance vector $\boldsymbol{\varepsilon}$, then LASSIE executes phase P_6.

Note that *Point 1* implicitly states that the system of ODEs is considered to be stiff, so that LASSIE automatically switches to phase P_5, where the BDF methods are used for the numerical integration. Phase P_4 is executed on the host.

Phase P_5. It implements the BDF methods, the most widely used implicit multi-step numerical integration algorithms [438]. LASSIE switches to this phase if and only if the RKF solutions $\mathbf{u}$ and $\mathbf{w}$ evaluated during phase P_4 are rejected as well as the RKF step-size h becomes smaller than ε_s.

Since the evaluation of the Jacobian matrix required by the BDF methods during each iteration is computationally expensive, LASSIE actually exploits the modified Newton-Raphson method [403]. Thus, the iteration matrix is evaluated once at the beginning of each step, based on the predicted value $\mathbf{x}^0$, and it is used for all the iterations during the current step. The linear system is solved by using the LU factorization method [30], which is a direct method supplied by the Nvidia CUDA Basic Linear Algebra Subroutines (cuBLAS) [314], a GPU-accelerated version of the standard BLAS library [248]. The Newton-Raphson method is iterated until the maximum number of iterations is reached, or a sufficiently accurate value is achieved (i.e., smaller than a user-defined tolerance value ε_{NR}). When this method ends, the state of the system is updated as $\mathbf{x} = \mathbf{x}^{i+1}$ (see Section 1.2.1 for more details). The workflow of phase P_5 is schematized in Figure 4.5.

Overall, phase P_5 is implemented by means of the following kernels:

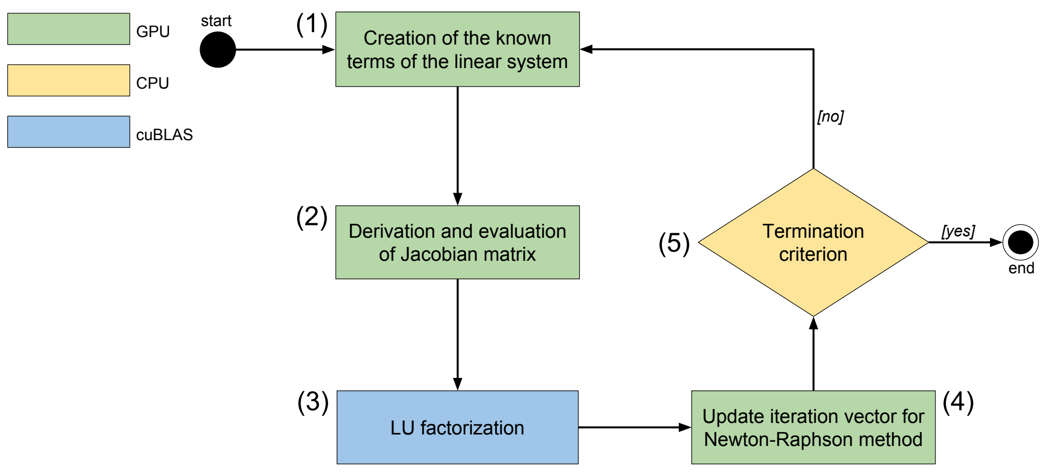

GPU
CPU
cuBLAS
start
(1)
Creation of the known terms of the linear system
(2)
Derivation and evaluation of Jacobian matrix
(3)
LU factorization
(4)
Update iteration vector for Newton-Raphson method
(5)
Termination criterion
[no]
[yes]
end

Fig. 4.5 Flowchart of phase P_5. (1) Each thread j, for $j = 0,\dots,N-1$, calculates the j-th known term of the linear system given in Equation 1.14, which is obtained from the system of non-linear equations given in Equation 1.12; (2) each thread j derives the j-th row of the Jacobian matrix and evaluates it on the current state of the system $\mathbf{x}$; (3) the linear system is solved by exploiting the LU factorization method supplied by cuBLAS library; (4) the solution of the linear system is used to update the iteration vector for the execution of the Newton-Raphson method; (5) steps 1 to 4 are iterated until the termination criterion is satisfied, that is, a maximum number of iterations is reached, or a value smaller than a fixed tolerance value ε_{NR} is achieved.

- **kernel $\mathbf{K_{11}}$**: each thread j, for $j = 0,\dots,N-1$, derives the j-th row of the Jacobian matrix and evaluates it on the current state of the system $\mathbf{x}$;
- **kernel $\mathbf{K_{12}}$**: the Jacobian matrix is transposed in order to exploit the LU factorization method;
- **kernels $\mathbf{K_{13}} - \mathbf{K_{18}}$**: based on the order q of the BDF, LASSIE invokes one of these kernels (i.e., **kernel $\mathbf{K_{13}}$** for $q = 1$, **kernel $\mathbf{K_{14}}$** for $q = 2$, ..., **kernel $\mathbf{K_{18}}$** for $q = 6$) to calculate the known terms of the linear system;
- **kernels $\mathbf{K_{19}} - \mathbf{K_{24}}$**: each **kernel $\mathbf{K_{(18+q)}}$**, $q = 1,\dots,6$, performs the calculations of the q-th order BDF;
- **kernel $\mathbf{K_{25}}$**: it updates the iteration vector needed to execute the Newton-Raphson method.

Phase P_6. It is used to verify the termination criterion: if the maximum time t_{max} is reached, then the simulation ends. On the contrary, the execution iterates from phase P_2. This phase is executed on the host.

All the temporary results computed by LASSIE are stored on the GPU, since data transfers between the CPU and the GPU (and viceversa) are very time consuming. For the same reason, the output data (i.e., the concentration values of the molecular species sampled at fixed time instants) are transferred to the CPU as soon as the whole simulation is completed.

4.2.2 Results

The computational performance of LASSIE was compared against the Livermore Solver of Ordinary Differential Equations (LSODA) [341] ODE solver, which is generally considered one of the best ODE solver for deterministic simulations of biological systems, thanks to its capability of dealing with stiff and non-stiff systems. In particular, we exploited the LSODA implementation provided by SciPy library [219] (version 0.15.1), written in `C` language. LASSIE was run on a machine with a GPU Nvidia GeForce GTX Titan Z, based on the Kepler architecture and equipped with 2×15 SMs for a total of 5760 cores (clock 837 MHz) and a theoretical peak processing power of 1.3 TFLOPS in double precision. Instead, LSODA was run on GALILEO, a supercomputer created by the Italian consortium CINECA. GALILEO consists of 516 compute nodes, each one equipped with 2 CPUs octa-core Intel® Xeon® Haswell E5-2630 v3 (clock 2.40 GHz) for a total of 8256 cores, and 128 GB of RAM. Each CPU is capable of about 300 GFLOPS in double precision. In our tests, we exploited one node with 120 GB of RAM distributed over 5 cores.

The computational performance was evaluated by simulating a set of synthetic RBMs of increasing size, that is, having a number of reactions and species $M \times N$ arbitrarily chosen in the range from 64×64 to 8192×8192. The models were generated considering the methodology used in [234, 307], which was modified in order to randomly sample the initial concentration of each species with a uniform distribution in the range $[0, 1)$, and the kinetic constant of each reaction with a logarithmic distribution in the range $[10^{-8}, 1)$.

For each model size $M \times N$, we generated and simulated 30 different synthetic RBMs to the aim of measuring the average running time of both LASSIE and LSODA. The simulation of each RBM was performed multiple times, using different settings for the sampling of the time-series. Specifically, in each repetition, we saved either $10, 50, 100, 500$ or 1000 samples of the system dynamics of all chemical species, at regular intervals. All simulations were halted at time $t_{max} = 50$ (arbitrary units).

All simulations were executed—independently from the size of the model and the number of samples saved—by setting the following parameters of LASSIE:

- tolerance of RKF method $\varepsilon_j = 10^{-12}$, $j = 1, \dots, N$;
- first–order BDF method ($q = 1$);

- BDF integration step $h = 0.1$;
- tolerance of Newton–Raphson method $\varepsilon_{NR} = 10^{-6}$;
- maximum number of iterations allowed during each call of the Newton-Raphson method $max_{it} = 10^4$;
- initial integration step of RKF method equal to 10^{-3};
- tolerance value to switch between RKF method and Backward Euler method (EM) $\varepsilon_s = 10^{-6}$.

The following parameters of LSODA were used to run the simulations:

- relative tolerance $\varepsilon_r = 10^{-6}$;
- absolute tolerance $\varepsilon_a = 10^{-12}$;
- maximum number of internal steps equal to 10^4.

Tables 4.1 and 4.2 report the values of the average running times (given in seconds) of LSODA and LASSIE, required for the execution of each set of 30 different synthetic RBMs of size $M \times N$, each time considering $10, 50, 100, 500, 1000$ samples of the system dynamics of all chemical species. The speed-up values achieved by LASSIE with respect to LSODA are given in Tables 4.1 and 4.2 and graphically represented in Figure 4.6, for each tested case; note that when the speed-up value is greater than one, LASSIE is faster than LSODA, and vice versa. The break-even (blue line in Figure 4.6) between the performances of LASSIE and LSODA is observed when the number of reactions and chemical species is between 128 and 256. Specifically, in the case of 256×256 model size and 10 samples, the running time of LSODA is almost twice with respect to LASSIE: 1.28 seconds vs. 0.67 seconds. In particular, we emphasize that the execution of the simulations for models characterized by 4096 reactions and 4096 species with 10 samples takes, on average, 249.8 seconds with LSODA and just 2.71 seconds with LASSIE, resulting in around $92\times$ speed-up. Furthermore, LASSIE allows for the simulation of large-scale models (e.g., 8192×8192) thanks to its smaller memory footprint with respect to LSODA, taking just 14.13 seconds to simulate the model characterized by 8192 reactions and 8192 species with 10 samples. Conversely, the version of LSODA implemented in SciPy library has a high memory footprint that does not allow for simulating models of this size on GALILEO, the supercomputer employed to perform the simulations.

Figure 4.6 also points out that the number of samples of the dynamics affects the performance of LASSIE, due to the different number of accesses to the high-latency global

Table 4.1 Average running time (in seconds) of LSODA and LASSIE – and corresponding speed-up value – required for the execution of the set of 30 synthetic RBMs of size $M \times N$ (with $M = N$), considering 10, 50 and 100 samples of the dynamics of all chemical species.

	10 samples			**50 samples**			**100 samples**		
$M \times N$	*LSODA*	*LASSIE*	*speed-up*	*LSODA*	*LASSIE*	*speed-up*	*LSODA*	*LASSIE*	*speed-up*
64 × 64	0.257	0.519	0.495	0.288	0.541	0.532	0.220	0.557	0.395
128 × 128	0.393	0.613	0.641	0.473	0.635	0.745	0.507	0.644	0.787
256 × 256	1.277	0.669	1.909	1.486	0.696	2.135	1.293	0.727	1.779
512 × 512	4.313	0.792	5.446	4.629	0.841	5.504	4.559	0.915	4.982
1024 × 1024	15.753	0.955	16.495	15.707	1.056	14.874	16.201	1.201	13.490
2048 × 2048	61.824	1.662	37.199	61.748	1.987	31.076	61.762	2.397	25.766
4096 × 4096	249.839	2.713	**92.090**	248.234	4.571	54,306	249.422	5.665	44.029
8192 × 8192	*NA*	14.134	*NA*	*NA*	26.051	*NA*	*NA*	38.058	*NA*

Table 4.2 Average running time (in seconds) of LSODA and LASSIE – and corresponding speed-up value – required for the execution of the set of 30 synthetic RBMs of size $M \times N$ (with $M = N$), considering 500 and 1000 samples of the dynamics of all chemical species.

	500 samples			**1000 samples**		
$M \times N$	*LSODA*	*LASSIE*	*speed-up*	*LSODA*	*LASSIE*	*speed-up*
64 × 64	0.307	0.665	0.462	0.303	0.839	0.361
128 × 128	0.674	0.786	0.858	0.498	0.958	0.520
256 × 256	1.319	0.905	1.456	1.277	1.122	1.138
512 × 512	4.300	1.215	3.539	4.669	1.526	3.060
1024 × 1024	15.982	1.809	8.835	16.647	2.407	6.916
2048 × 2048	62.307	3.721	14.745	62.742	5.479	11.451
4096 × 4096	249.546	12.407	20.113	254.416	17.393	14.627
8192 × 8192	*NA*	101.91	*NA*	*NA*	129.755	*NA*

memory. For instance, the speed-up achieved with the model characterized by 4096 reactions and 4096 species decreases to 14.6× with 1000 samples, meaning that the simulations with 1000 samples are around 6× slower than the simulations with 10 samples. In models characterized by 2048 reactions and 2048 species, the speed-up obtained with 10 samples (37.2×) is around 3× larger compared to the one achieved with 1000 samples (11.4×), while in models characterized by 1024 reactions and 1024 species, the speed-up obtained with 10 samples (16.5×) is around 2× larger compared to the one achieved with 1000 samples (6.9×). Finally, Figure 4.7 shows that the running time of LASSIE increases with the number of samples, while LSODA is characterized by an almost constant running time, irrespective of the number of samples. It is worth noting that CPU-bound ODE solvers, like this version of LSODA, can be more efficient in the case of small-scale models. This is due to two concomitant circumstances. On the one hand, the clock frequency of CPUs is higher than the clock frequency of GPU (2.4 GHz with respect to 837 MHz, in the case of the hardware used to execute our tests). On the other hand, the communication and synchronization between threads can introduce a significant overhead, which is mitigated only when the calculations are distributed over a relevant number of threads; therefore, LASSIE becomes profitable for

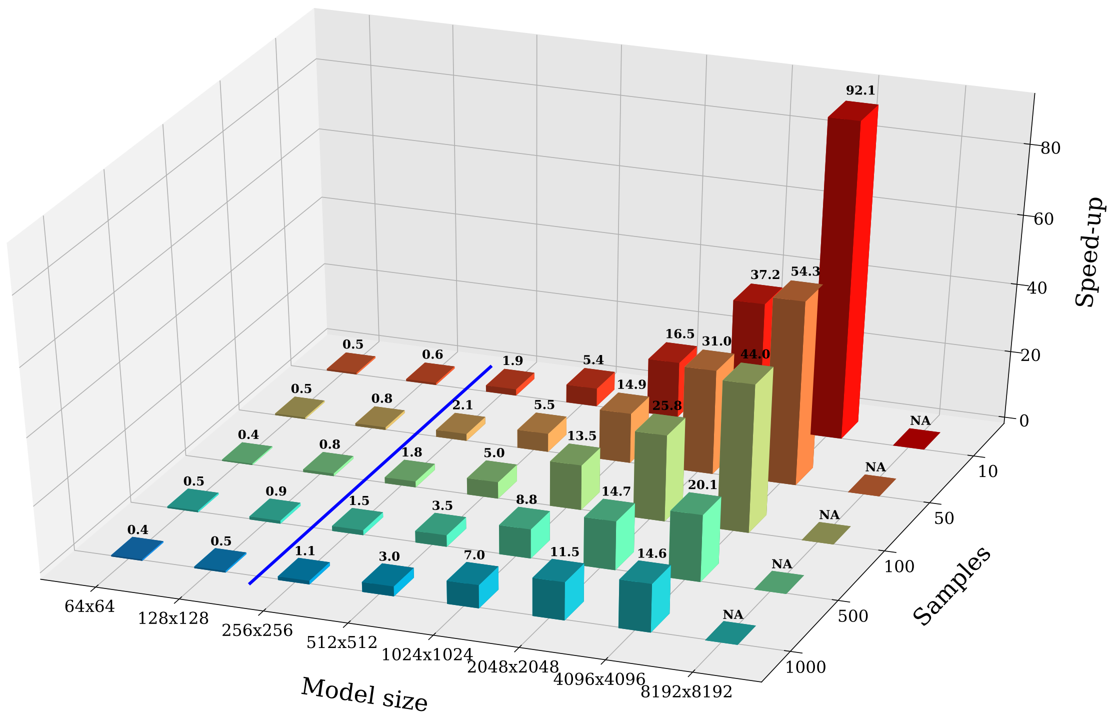


Fig. 4.6 speed-up values (z-axis) achieved by LASSIE with respect to LSODA for the simulation of synthetic models of increasing size, having a number of reactions and of species $M \times N$ (x-axis), with $M = N$, and characterized by an increasing number of sampling time instants of the system dynamics (y-axis). When the value of the speed-up is greater than one, LASSIE is faster than LSODA and vice versa.

medium/large-scale models characterized by hundreds of species. Notably, the bigger the model, the greater the speed-up.

As an additional test, we investigated whether the relationship between the number of reactions and the number of species could affect the overall performance of LASSIE. As the number of chemical species corresponds to the number of ODEs, the length of each ODE is roughly proportional to the number of reactions. Since GPUs have a lower clock frequency than CPUs (e.g., in the case of the hardware used for the tests, 837 MHz with respect to 2.4 GHz, respectively), each GPU core is slower than the CPU core to perform a single instruction [1]. For this reason, in order to obtain the highest performance, the calculations on the GPU should be spread across threads as much as possible, while the number of operations performed by each thread should be reduced.

[1]The advances in GPU technology will progressively reduce this gap. As a matter of fact, Nvidia is continually working to increase both the clock frequency and the band memory, two features that, theoretically, are expected to increase the performance of LASSIE.

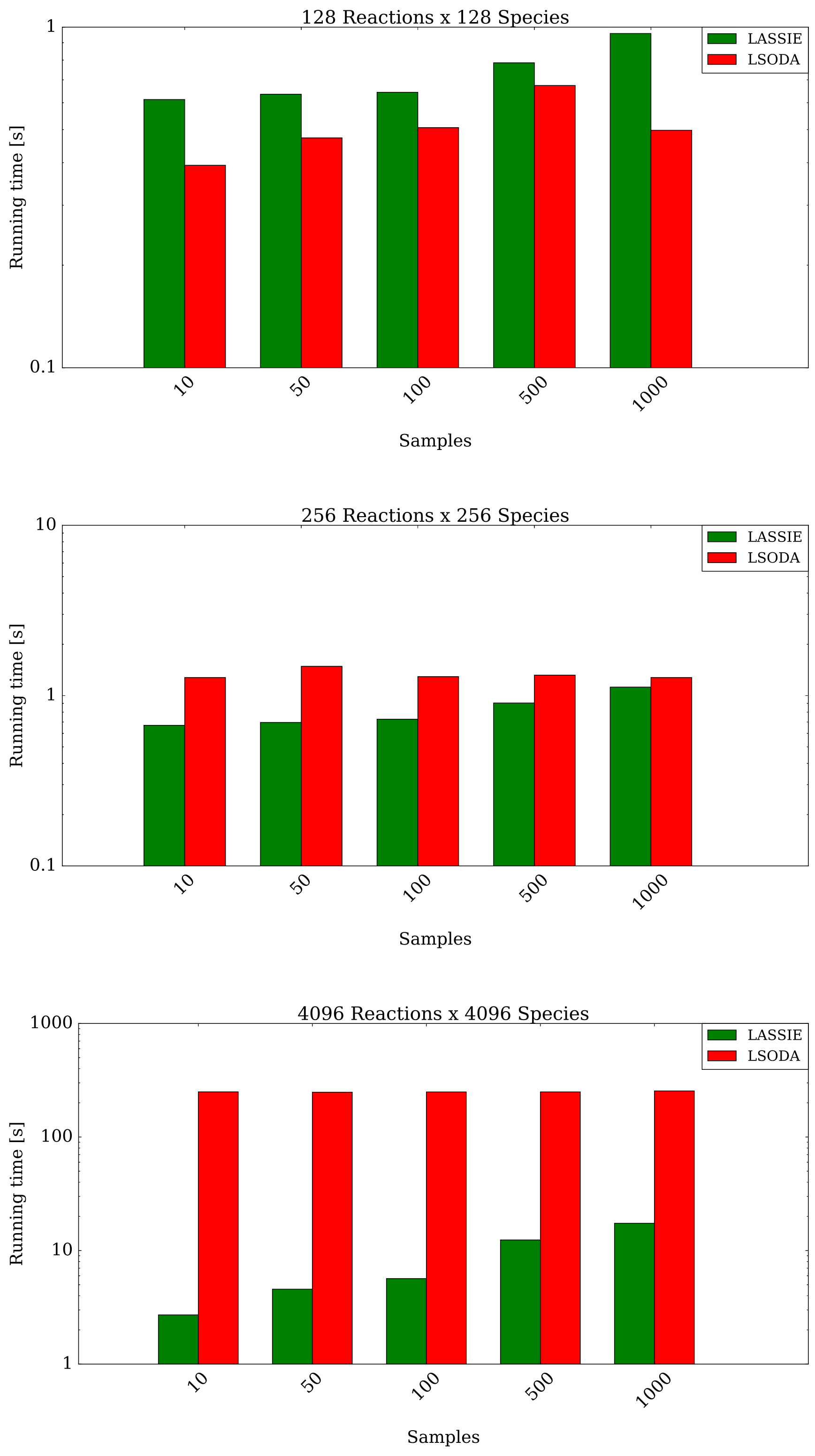


Fig. 4.7 Comparison between the average running time required by LASSIE (green bars) and LSODA (red bars) to simulate 30 instances of models characterized by 128 reactions and 128 species (top), 256 reactions and 256 species (middle), 4096 reactions and 4096 species (bottom), saving different numbers of sampling time instants of the dynamics. Note that the y-axes are in logarithmic scale.

Table 4.3 Average running time (in seconds) of LASSIE required for the execution of a set of 30 synthetic RBMs of size $M \times N$ (with $M \neq N$), considering 10, 50, 100, 500, 1000 samples of the dynamics of all chemical species.

	10 samples	**50 samples**	**100 samples**	**500 samples**	**1000 samples**
$M \times N$	*LASSIE*	*LASSIE*	*LASSIE*	*LASSIE*	*LASSIE*
171 × 512	0.589	0.623	0.662	0.900	1.149
512 × 171	1.032	1.062	1.087	1.318	1.540
341 × 1024	0.682	0.751	0.828	1.285	1.726
1024 × 341	1.119	1.179	1.220	1.597	1.895
512 × 1024	0.759	0.828	0.941	1.419	1.949
1024 × 512	1.053	1.112	1.192	1.624	1.999
683 × 2048	0.876	1.064	1.310	2.442	3.533
2048 × 683	1.389	1.512	1.664	2.406	3.029
1024 × 2048	1.002	1.201	1.508	2.858	3.889
2048 × 1024	1.317	1.444	1.613	2.614	3.281

Indeed, as reported in Table 4.3 and shown in Figure 4.8, when the number of chemical species involved in a model is greater than the number of reactions, LASSIE achieves better performance than those obtained in the case of models with a number of chemical species smaller than the number of reactions. For instance, considering the models with $M \times N$ equal to 171×512, the running time of LASSIE is smaller than in the case of the models with size 512×171, irrespective of the number of samples of the system dynamics, thanks to the higher number of threads that are concurrently launched on the GPU in the first case.

This is in general valid in all cases with the exception of the models characterized by 2048 chemical species with 500 and 1000 samples of the system dynamics. Here, the average running time of LASSIE is greater than in the case of models with 2048 reactions, since the required number of accesses to the high-latency global memory of the GPU impairs the performance of the simulations.

In order to assess the scalability of LASSIE, and of CUDA applications in general, we executed additional tests on different GPUs. Figure 4.9 shows a comparison of the performance of LASSIE using three different GPU models (described in Table 4.4): a notebook video card (Nvidia GeForce 960M, red bars), the Nvidia GeForce GTX Titan Z used for the previous tests (green bars), and a Tesla-class GPU (the Nvidia K20c, blue bars). To compare the speed-up provided by these GPUs we generated 30 different synthetic models (characterized by size $M \times N$ equal to 1024×1024, 2048×2048 and 4096×4096) and calculated the average running time. Our results highlight the importance of two distinct factors on the performance of LASSIE: the GPU clock frequency and the amount of available resources (in this case, the cores). As a matter of fact, despite the lower amount of CUDA cores, the GeForce 960M turns out to be competitive on models of moderately large size, thanks to its higher clock rate, with respect to the Titan Z and the K20c. When the ODEs largely outnumber the available cores (e.g., for 4096 reactions and chemical species), the

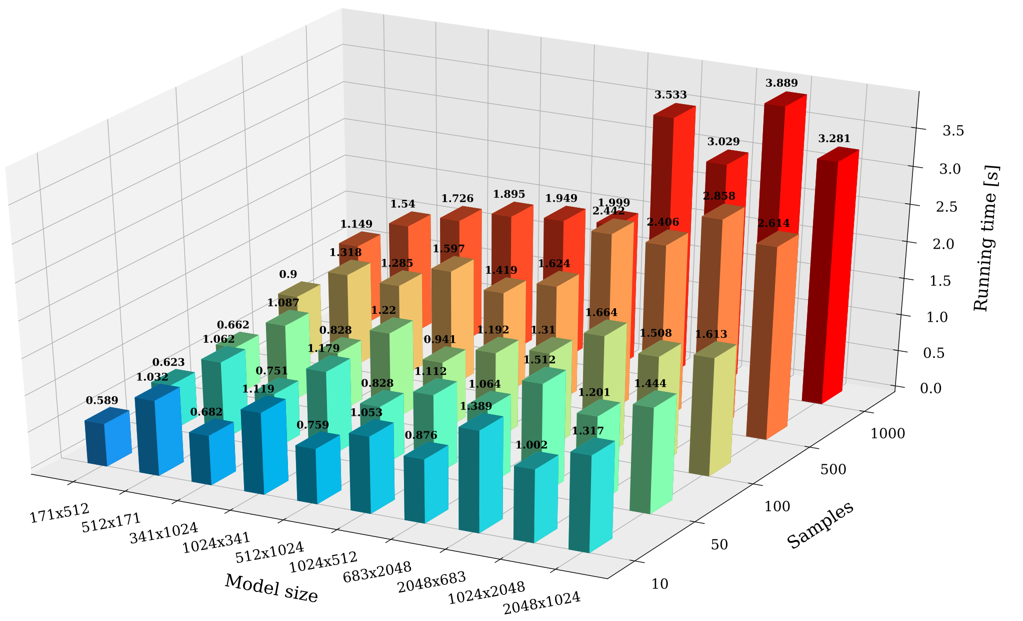


Fig. 4.8 Running time (z-axis) of LASSIE for the simulation of synthetic models of increasing size, having a number of reactions and of species $M \times N$ (x-axis), with $M \neq N$, and characterized by an increasing number of sampling time instants of the system dynamics (y-axis).

GeForce 960M is no longer competitive. This is an example of transparent scalability of CUDA applications: the threads are automatically distributed over the available cores, improving the overall performance, without any user intervention. Moreover, as described in Section 3.4.1, threads are organized in blocks that are scheduled on the available SMs. Thanks to this characteristic, when the overall number of threads outnumbers the available cores, CUDA automatically creates a queue of blocks that are scheduled on the SMs as soon as they become available for computation. Thus, LASSIE can, in principle, simulate any model on any GPU, as long as there is enough memory to store the data structures. The Tesla K20c is characterized by a large amount of cores that, in the case of 1024×1024 models, are fully exploited only during the simulation of the stiff parts of the dynamics. For the remaining parts of the simulation, half of its cores are actually used for computation with a slower clock rate with respect to the clock rate of the GeForce GPUs. Moreover, Tesla cards exploit the Error Correcting Code (ECC) on memories, ensuring additional checks of correctness to the data against potential corruption from electrical or magnetic interference, at the price of a significant overhead [468]. The ECC was enabled during all tests, partly explaining the reduced performance of the Tesla K20c on very large-scale models with respect to the Titan Z.

Table 4.4 Nvidia GPUs used to assess the scalability of LASSIE.

	GeForce GTX 960M	**GeForce GTX Titan Z**	**Tesla K20c**
Global memory	4 GB	6 GB	5 GB
Number of SMs	5	15	13
CUDA cores per SM	128	192	192
Total number of CUDA cores	640	2880	2496
Base clock	1.2 GHz	876 MHz	706 MHz

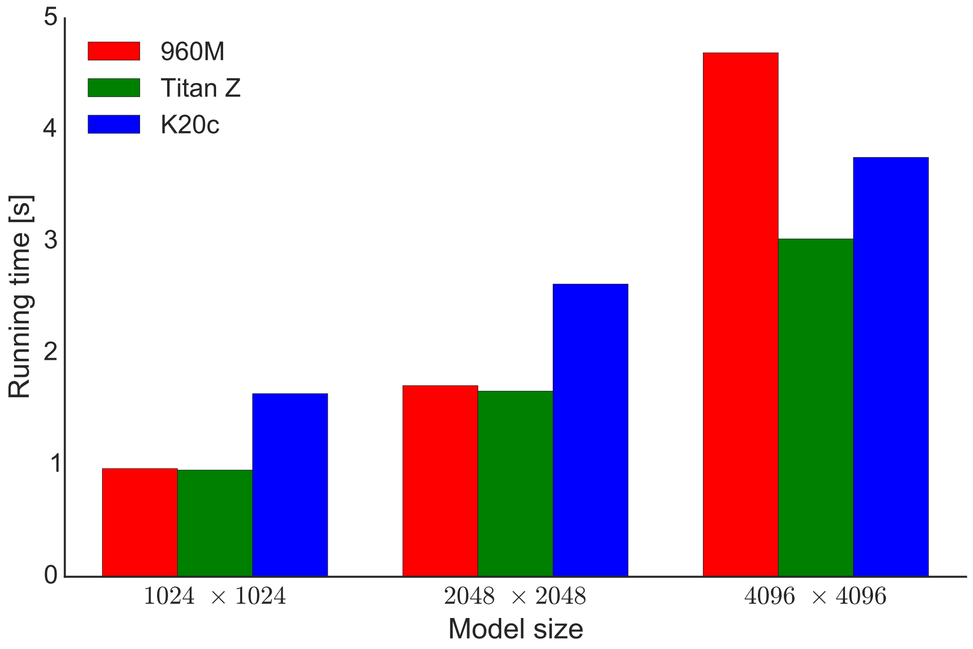


Fig. 4.9 Comparison of the average running times for the simulation of 30 synthetic models characterized by three different sizes, executed with different GPUs: a notebook GPU Nvidia GeForce 960M (red bars); a Nvidia GeForce GTX Titan Z (green bars); a Tesla-class GPU Nvidia K20c (blue bars).

Finally, we assessed the accuracy of LASSIE by simulating the dynamics of the model of the Ras/cAMP/PKA pathway in yeast (see Appendix A.5), and comparing the outcome of LASSIE with the result of the simulation performed with LSODA. We also investigated the influence of LASSIE parameters (e.g., tolerance values) on the running times and quality of the simulated dynamics by exploiting a model representing a chain of isomerizations.

The simulation accuracy was tested by comparing the dynamics of four pivotal molecular species involved in the RBM of Ras/cAMP/PKA pathway [75], which consists in 34 reactions among 30 molecular species and is characterized by stiffness. Figure 4.10 shows that the

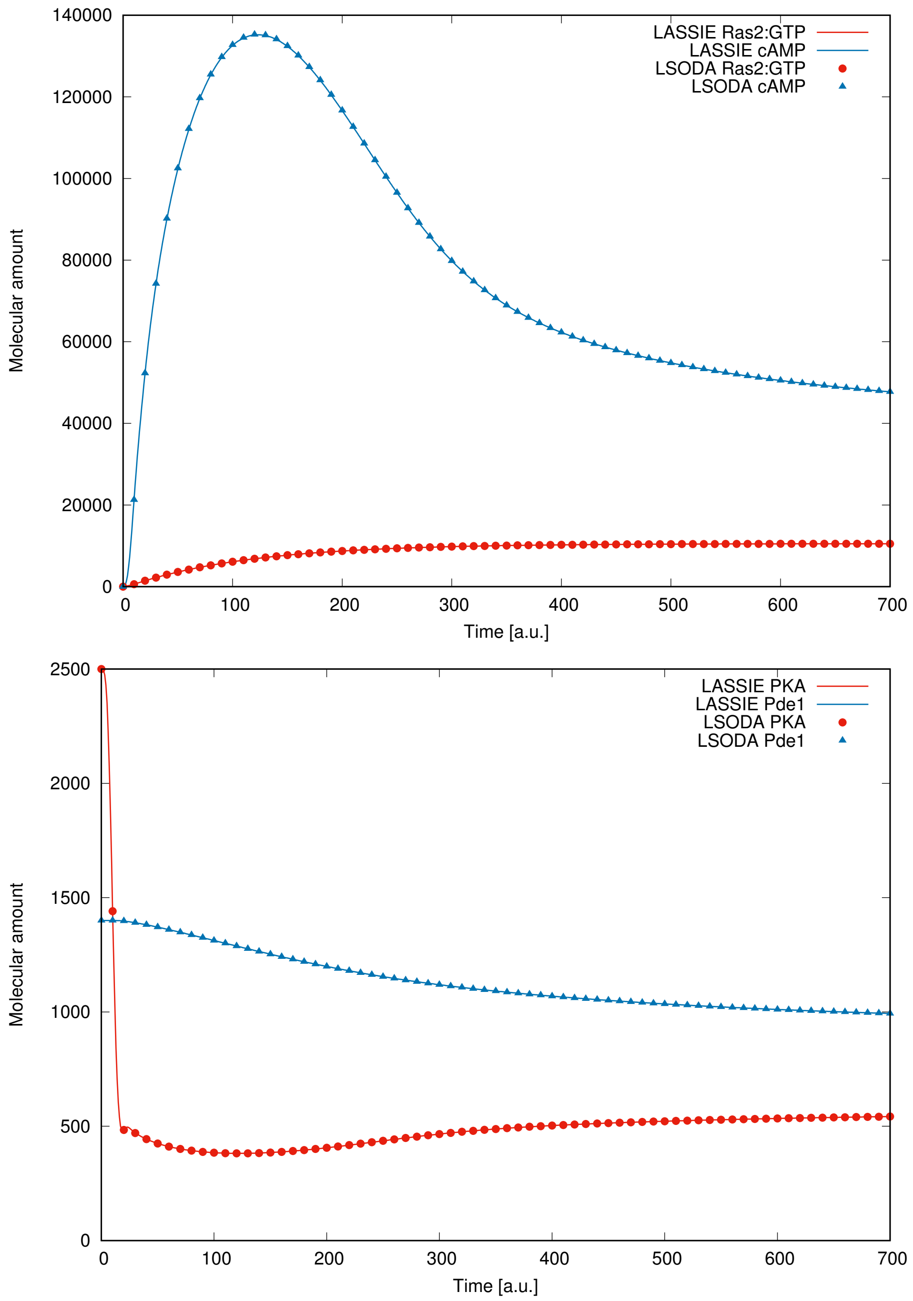


Fig. 4.10 Comparison of the dynamics of the molecular species Ras2:GTP and cAMP (top), PKA and Pde1 (bottom) of the model of the Ras/cAMP/PKA signaling pathway in yeast, obtained by running LASSIE (solid lines) and LSODA (dots).

dynamics of species Ras2:GTP and cAMP (top), PKA and Pde1 (bottom) perfectly overlap, proving the accuracy of LASSIE.

To assess the robustness of the parameterization of LASSIE (e.g., tolerance values set in all results presented above), we performed an additional batch of tests by changing one

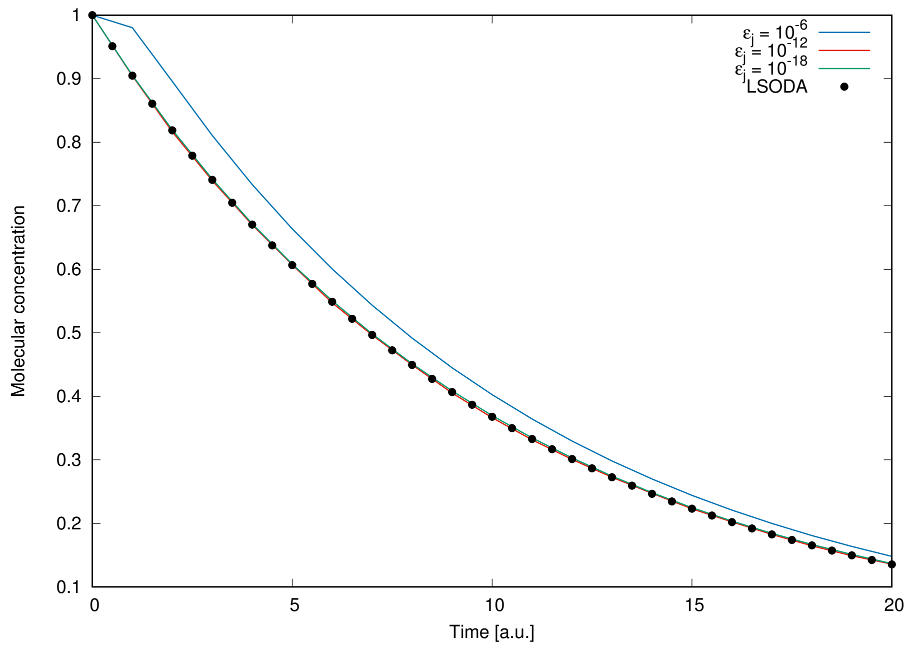


Fig. 4.11 Comparison of the dynamics of species S_1 of the chain of isomerizations model obtained with LSODA (dots) and using different values of the tolerance ε_j of RKF method exploited by LASSIE (solid lines).

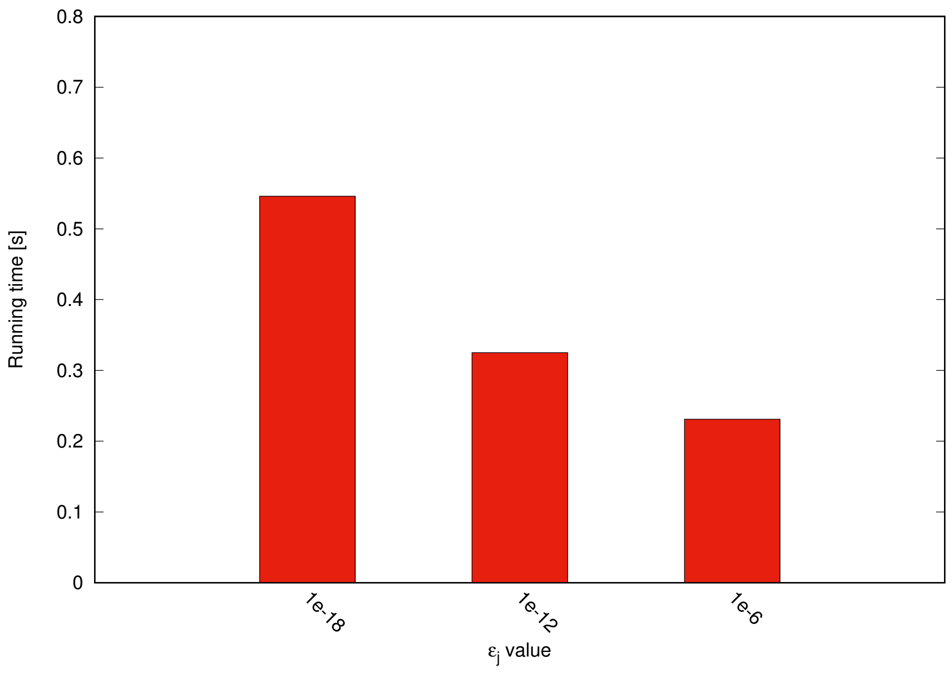


Fig. 4.12 Running time required by LASSIE for the simulation of the chain of isomerizations model using different values of the RKF method tolerance ε_j.

parameter at a time, and measuring the effect of the corresponding perturbations on the quality of the output dynamics and on the running time. For the execution of these tests, we considered a model characterized by stiffness, representing a chain of isomerizations in which a chemical species S_i (where $i = 1, \dots, N-1$) undergoes a cascade of modifications: $S_i \longrightarrow S_{i+1}$. To be more precise, this model consists in 31 reactions among 32 chemical species, where the kinetic constant of each reaction is set to 0.1, and the initial concentration of species S_1 is equal to 1 (the initial concentration of all other species is equal to 0). The variation of the tolerance value of the Newton–Raphson method, as well as the tolerance value used to switch between RKF and Backward EM, have no relevant effect on the quality of the output dynamics and on the running time of LASSIE. On the contrary, the variation of the tolerance value of RKF method, which is generally set as default to $\varepsilon_j = 10^{-12}$, can have a substantial effect on the output dynamics of LASSIE, as shown in Figure 4.11. Taking into account the chain of isomerizations model, the dynamics is correctly reproduced in the case of $\varepsilon_j = 10^{-12}$ and $\varepsilon_j = 10^{-18}$, while in the case of $\varepsilon_j = 10^{-6}$, the dynamics does not overlap the simulation outcome obtained by running LSODA. In addition, as the value of ε_j decreases, the running time of LASSIE increases, as shown in Figure 4.12. However, considering that a value of $\varepsilon_j = 10^{-12}$ is sufficient to ensure a correct replication of the model dynamics, it is unnecessary to further reduce this tolerance value.

4.2.3 Conclusion

LASSIE is a fully automatic simulator in which the system of ODEs corresponding to an RBM is automatically determined according to the mass-action kinetics, making LASSIE usable without any prior knowledge about ODE modeling and integration. The mass-action law allows for obtaining a first order ODE for each species appearing in the model: it is worth noting that this ODE is a *polynomial function* that describes how the concentration of that species changes in time, according to all the reactions where it appears either as reactant or product [455]. The presence of polynomial functions simplifies the symbolic derivation that is needed to calculate the Jacobian matrix associated with the system of ODEs and exploited by implicit integration methods. In addition, polynomials can be efficiently encoded in the memory and parsed GPU-side. As a result, all GPU threads can perform the same task (i.e., polynomial decoding and evaluation), strongly reducing the warp divergence and the consequent stalling of threads due to serialization, a circumstance that would instead happen if each thread calculated an ODE characterized by an arbitrary kinetics.

In order to assess the computational performance of LASSIE, we performed a set of simulation tests using synthetic RBMs of increasing size, and we compared LASSIE running time with respect to the LSODA ODE solver running on CPU. Despite the improvement of efficiency granted by LSODA, the numerical integration of the system of ODEs can become excessively burdensome when the numbers of reactions and molecular species increase. LASSIE overcomes this limitation by distributing over thousands of GPU cores all the calculations required by the numerical integration methods it embeds, therefore paving the way for fast simulations of large-scale and stiff models of cellular processes. LASSIE execution flow was partitioned into 25 CUDA kernels, overall distributing the calculations over the available cores in order to fully exploit the massive parallel capabilities of modern GPUs, therefore achieving a relevant reduction of the running time in case of large-scale models. One interesting feature of GPUs is that they can have different characteristics, both in terms of resources (e.g., amount of high performance memories, number of cores) and computing power (e.g., clock rate). For instance, kernel performance transparently scale on different GPUs, since they automatically leverage the additional resources offered by the latest architectures, a characteristic known as transparent scalability. We assessed this feature testing LASSIE on different GPUs and showing how the number of available SMs effect the performance. Indeed, the higher the number of available SMs, the lower the required running time.

As a final remark, we highlight that a fair comparison of GPUs and CPUs is a difficult task, in general, due to their deep architectural differences. The theoretical peak performance of both architectures are difficult to achieve: indeed, developers must implement code to the aim of maximizing the parallelism and the occupancy of the multi-processors, adhering as much as possible to the underlying SIMD computational model in the case of the GPU and exploiting vector instructions in the case of the CPU. However, as explained in Section 3.4.1, GPUs allow for the temporary divergence of the execution flow of threads. When this situation occurs, some threads get stalled waiting for reconvergence. This mechanism provides the programmer with a certain degree of freedom to abandon the SIMD paradigm, but at the same time it can potentially lead to the complete serialization of the execution affecting the overall performance. We also highlight that the usage of registers and shared memory influences the occupancy of the GPU, as these resources are scarce on each SM. All these circumstances can prevent the achievement of the peak computational power of a GPU. To this aim, we developed kernels that maximize the parallelism and the occupancy of the SMs avoiding threads divergence as much as possible. Moreover, we optimized data structures to store the matrices $\mathbf{A}$ and $\mathbf{H}$ that encode the system of ODEs, and CUDA vector types that allow us to increase the memory throughput and to reduce the number of memory accesses, all precautions that explain the performance boost achieved with LASSIE.

4.3 FiCoS

In the previous section we discussed LASSIE, the first fine-grained deterministic simulator in literature. In order to fully exploit the parallelism provided by modern GPU, here we present FiCoS, an efficient deterministic simulator accelerated on GPU by exploiting both the fine- and coarse-grained parallelization strategies: the former is used to distribute the calculations required by every ODE over multiple GPU cores, while the latter is exploited to perform a massive number of simulations in a parallel fashion. As shown in Figure 4.1, FiCoS is the first deterministic simulator that takes advantage of both parallelization strategies, filling the gap in the combined fine- and coarse-grained deterministic simulation. Regarding the fine-grained strategy, FiCoS takes inspiration from LASSIE, but at the same time it is based on more efficient numerical integration methods, which allow for strongly reducing the computational effort required by deterministic simulations. As in Variable-coefficient ODE solver (VODE) [53] (see Section 1.2.1), FiCoS exploits an heuristic to determine if the system is stiff and which integration method is the most appropriate. It implements two integration methods belonging to the Runge-Kutta family: (*i*) the Dormand–Prince (DOPRI) method [116, 115, 176] for stiffness-free systems; (*ii*) the Radau IIA method [177, 178] when the system is stiff. To be more precise, we used DOPRI5 and RADAU5, which are explicit and implicit adaptive Runge-Kutta methods of order 5, respectively, capable of varying the integration step-size during the resolution of the system of ODEs (see Section 1.2.1).

In order to compare the computational performance of FiCoS against CPU-based (i.e., LSODA [341] and VODE [53]) and GPU-based (i.e., cupSODA [306] and LASSIE) ODE solvers, we carried out different batches of simulations using (*i*) a set of symmetric synthetic models of increasing size, ranging from 64 to 800 species and reactions, to evaluate the impact of the models size on the computational performance; (*ii*) a set of asymmetric synthetic models of increasing size, ranging from 21 species (64 reactions) to 267 species (800 reactions) and viceversa, to evaluate in which way the number of species and reactions affects the performance of the GPU-powered simulators; (*iii*) a real model of the Autophagy/Translation switch based on the mutual inhibition of MTORC1 and ULK1 [422], characterized by 173 molecular species and 6581 reactions.

For each synthetic model, we performed an increasing number of parallel simulations (up to 2048) and calculated both the integration time and the overall simulation time. The *integration time* indicates the running time spent by the numerical integration algorithms to solve the system of ODEs, while the *simulation time* is the overall running time required to perform a simulation, including the I/O operations (i.e., reading and writing operations). In the case of multiple simulations, both the integration time and the simulation time were calculated by summing up the integration time and the simulation time required by each simulation. As a case study, we then performed a bidimensional Parameter Sweep Analysis [11, 85] (PSA-2D) varying two parameters of the Autophagy/Translation model to identify the initial conditions and reaction rates that allow for obtaining an oscillatory regime of the dynamics. In this case FiCoS was able to perform 36864 simulations in 24 hours, while LSODA and VODE completed only 2090 and 1363 simulations, respectively, in the same time. This result shows that FiCoS can be effectively coupled with the computational methods used to calibrate mathematical models of complex (large-scale) biochemical networks. Considering the achieved results, FiCoS was also exploited to perform an in-depth analysis of an RBM describing the Treg-Teff cross regulation in relapsing-remitting multiple sclerosis [31].

4.3.1 GPU implementation

As a first step, starting from the RBM given as input, FiCoS automatically generates the system of ODEs, according to Equation 1.4, and encodes both the matrices $\mathbf{H} = (\mathbf{B} - \mathbf{A})^T$ and $\mathbf{A}$ as two arrays of `short2` CUDA vector type, named $\mathbf{H_v}$ and $\mathbf{A_v}$, respectively. Note that we exploited the `short2` CUDA vector type because it is 4-aligned in the memory, meaning that a single instruction to fetch a whole entry is required. We used these data structures to compress and store the matrices $\mathbf{A}$ and $\mathbf{H}$, which are highly sparse, removing all zero elements to save memory, and at the same time avoiding unnecessary readings from the global memory of the GPU. The following strategy is applied to generate the CUDA vector type $\mathbf{H_v}$ and $\mathbf{A_v}$:

- for each non-zero element h_{ji} of $\mathbf{H}$, with $i = 1, \ldots, M$ and $j = 1, \ldots, N$, we stored into the `.x` and `.y` components of $\mathbf{H_v}$ the values i and h_{ji}, respectively. Notice that N corresponds to the number of species, while M corresponds to the number of reactions in the RBM;

- for each non-zero element a_{ij} of $\mathbf{A}$, with $i = 1, \ldots, M$ and $j = 1, \ldots, N$, we stored into the `.x` and `.y` components of $\mathbf{A_v}$ the values j and a_{ij}, respectively.

Besides these two data structures, we use two additional arrays of `int`, named $\mathbf{O_H}$ and $\mathbf{O_A}$, to parse $\mathbf{H_v}$ and $\mathbf{A_v}$ inside the GPU. $\mathbf{O_H}$ and $\mathbf{O_A}$ store the offsets to correctly fetch the entries of $\mathbf{H_v}$ and $\mathbf{A_v}$, respectively. Each thread j, with $j = 0, \ldots, N-1$, reads the j and $j+1$ elements of $\mathbf{O_H}$, whose values indicate the first index and the last index (minus 1) to access the $\mathbf{H_v}$ structure. Similarly, $\mathbf{O_A}$ stores the indeces that allow the threads to correctly decode the $\mathbf{A_v}$ structure. Finally, an array $\mathbf{K}$ of type `double` is used to store the values of the kinetic constants. An example of these data structures and their decoding is depicted in Figure

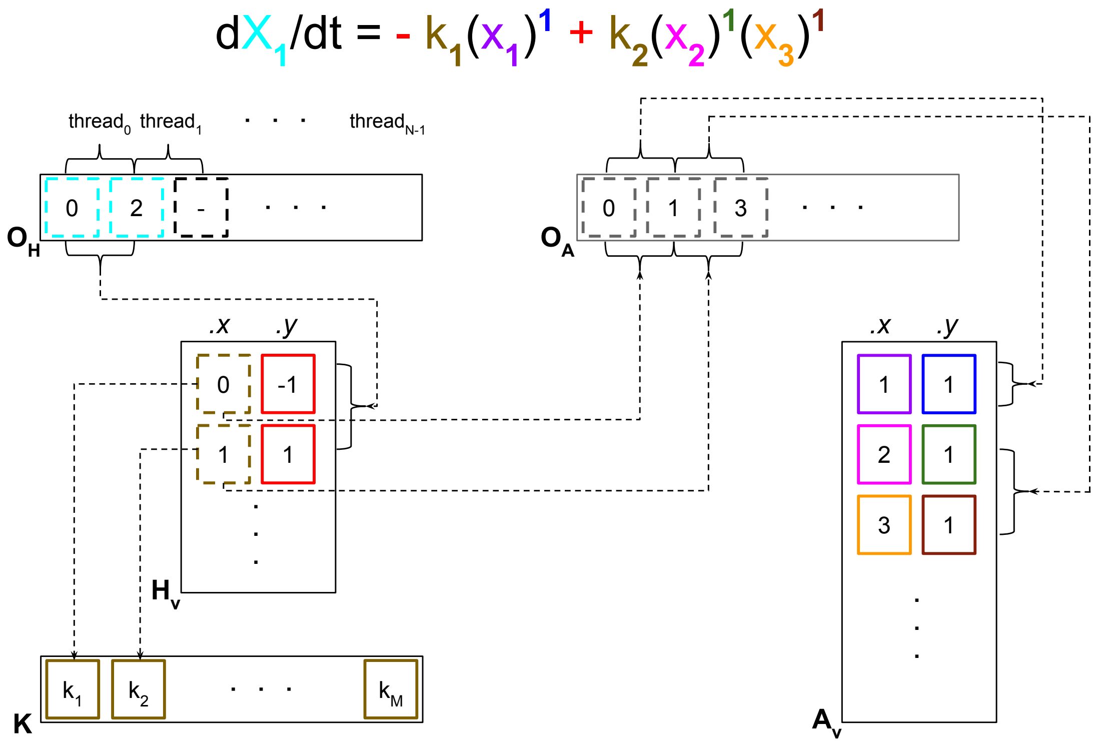


Fig. 4.13 Example of matrix encoding to automatically generate an ODE using FiCoS.

4.13, where the monomials composing the polynomial function that describes the ODE of the species x_1, at the top of the figure, are encoded in $\mathbf{O_H}$, $\mathbf{O_A}$, $\mathbf{H_v}$, $\mathbf{A_v}$ and $\mathbf{K}$. The ODE is automatically generated by using the data structure components with solid borders; the colors used to highlight the terms composing the ODE are also used in the data structure components. Each thread j, with $j = 0, \ldots, N-1$, reads the j and $j+1$ elements of $\mathbf{O_H}$ (denoted by the lightblue borders). Each thread j fetches the values stored in $\mathbf{H_v}$, starting from the row indicated by the value of the element j of $\mathbf{O_H}$ up to the row indicated by the value (minus 1, since this value is used as starting point for the thread $j+1$) of the element $j+1$ of $\mathbf{O_H}$. In this example, thread 0 reads the `.x` and `.y` components of the first two rows (i.e., rows 0 and 1) of $\mathbf{H_v}$. The `.x` component of each row of $\mathbf{H_v}$ (denoted by dark yellow) indicates both the position of the vector $\mathbf{K}$—which contains the values of the kinetic constants—and the first index of $\mathbf{O_A}$ where each thread j must read; the `.y` component (red borders) codifies both the sign and coefficient of the monomial. In this example, the `.x` and `.y` components of $\mathbf{H_v}$ codify the coefficients $-1k_1$ and $+1k_2$ of the first and the second term of the ODE, respectively. Afterwards, each thread j fetches the values encoded in $\mathbf{O_A}$ starting from the value stored in the position indicated by the index previously read up to value (minus 1) stored in the next position. The values stored in the `.x` (purple, magenta and orange borders) components of $\mathbf{A_v}$ are the indices of the species, while the `.y` (blue, dark green and dark red borders) components codify the stoichiometric coefficients. In this example, the row 0 of $\mathbf{A_v}$ contains the factor $(x_1)^1$, while the rows 1 and 2 allow the thread 0 to generate the factor $(x_2)^1(x_3)^1$. By so doing, the thread 0 is capable of reproducing the polynomial composing the ODE: $-k_1(x_1)^1 + k_2(x_2)^1(x_3)^1$.

Once the data structures encoding the system of ODEs are generated, FiCoS solves the systems of ODEs by means of the DOPRI5 method [116, 115, 176] in the absence of stiffness or the RADAU5 method [177, 178] when the system is stiff. We point out that our implementation was inspired by the source code of Blake Ashby, who ported the original `Fortran` code of Hairer and Wanner [177, 176] to `C++`.

Starting from an initial time instant t_0, the system of ODEs is integrated up to a given maximum simulation time t_{max}. The dynamics of the species are sampled and saved at specified time steps within the interval $[t_0, t_{max}]$ (these time steps should generally correspond to the sampling times of laboratory experiments).

The workflow of FiCoS can be summarized in 5 distinct phases, as depicted in Figure 4.14. Note that only phases P_1 and P_5 are executed by the host (light orange boxes in Figure 4.14), while the others are executed by the device (light green boxes in Figure 4.14). Phases P_2, P_3 and P_4 correspond to three different kernels (K_1, K_2, and K_3) developed to take advantage of the coarse-grained parallelization strategy. Moreover, these phases invoke other lightweight kernels, which were developed to fully leverage the parallel architecture of the modern GPU exploiting the dynamic parallelism [321] as fine-grained parallelization strategy for the implementation of the aforementioned numerical integration methods. By exploiting the dynamic parallelism, each thread belonging to the grid called by the host (named parent grid) can launch a novel grid (named child grid) composed of several threads. All child grids that are executed by the threads of the parent grid can be synchronized so that the parent threads can consume the output produced from the child threads without the involvement of the CPU.

We describe hereafter each phase with its CUDA kernels to explain our novel parallelization strategy that allowed FiCoS to achieve the relevant speed-up shown in Section 4.3.2.

Phase $\mathbf{P_1}$. This phase is executed by the host and implements the generation of the data structures encoding the system of ODEs, as discussed above.

Phase $\mathbf{P_2}$. This phase implements the estimation of the dominant eigenvalue ρ of the Jacobian matrix $\mathbf{J}$ by means of the norm bound of $\mathbf{J}$ itself. As norm bound we used the so-called maximum absolute row sum norm, defined as:

$$\|\mathbf{J}\|_\infty = \max_{1\leq i\leq M} \sum_{j=1}^{N} |J_{ij}|\,. \tag{4.1}$$

Note that the dominant eigenvalue ρ is equal to the spectral radius, which is defined as the largest absolute value of the eigenvalues of $\mathbf{J}$. Since the value ρ strictly depends on both the initial concentrations and the kinetic parameter values, ρ is calculated for each parallel simulation ϕ, with $\phi = 0,\dots,\Phi-1$, which must be performed. This phase is implemented using 2 different CUDA kernels:

- kernel K_1: each thread ϕ, with $\phi = 0,\dots,\Phi-1$, invokes the kernel K_{1_a} to evaluate the Jacobian matrix $\mathbf{J}$ using the ϕ-th parameterization and the ϕ-th set of initial concentrations as state of the system $\mathbf{x}_\phi$. As soon as $\mathbf{J}$ is calculated, the thread ϕ applies the Equation 4.1 to estimate the dominant eigenvalue ρ associated to the ϕ-th parameterization and the ϕ-th set of initial concentrations;

- kernel K_{1_a}: each thread j, with $j = 0,\dots,N-1$, calculates the values of the j-th row of $\mathbf{J}$ using the state of the system $\mathbf{x}_\phi$ and the parameterization given as input.

Note that each thread ϕ, with $\phi = 0,\dots,\Phi-1$, exploits the dynamic parallelism to call the kernel K_{1_a} launching a new grid of threads.

Phase P$_3$. It implements the DOPRI5 method [116, 115, 176], an explicit Runge-Kutta integration algorithm of order 5 with variable step-size and stiffness control, used by the threads whose dominant eigenvalue ρ is less than ε_ρ to solve the system of ODEs. As a matter of fact, the Φ threads are partitioned in two different sets: the former ($\mathcal{L}_{DOPRI}$) contains the threads whose dominant eigenvalue ρ is less than ε_ρ; the latter ($\mathcal{L}_{RADAU}$) contains the remaining threads. By simulating both synthetic and real RBMs, including some models that are characterized by stiffness (e.g., the Ras/cAMP/PKA signal transduction pathway in yeast, see Appendix A.5 for further details), we found an empirical value of ε_ρ equal to 500. Since this phase is performed before the phase that implements the RADAU5 method, if the DOPRI5 method fails in solving the system of ODEs characterized by some parameterization and set of initial concentrations ϕ (e.g., the system of ODEs is stiff), the ϕ-th thread is put in $\mathcal{L}_{RADAU}$ along with the threads related to the simulations characterized by $\varepsilon_\rho > 500$.

The DOPRI5 method relies on the following 12 kernels:

- kernel K_2: it is the main kernel implementing the DOPRI5 method. It is executed by each thread $d \in \mathcal{L}_{DOPRI}$, which uses its own parameterization as well as its own set of initial concentrations;

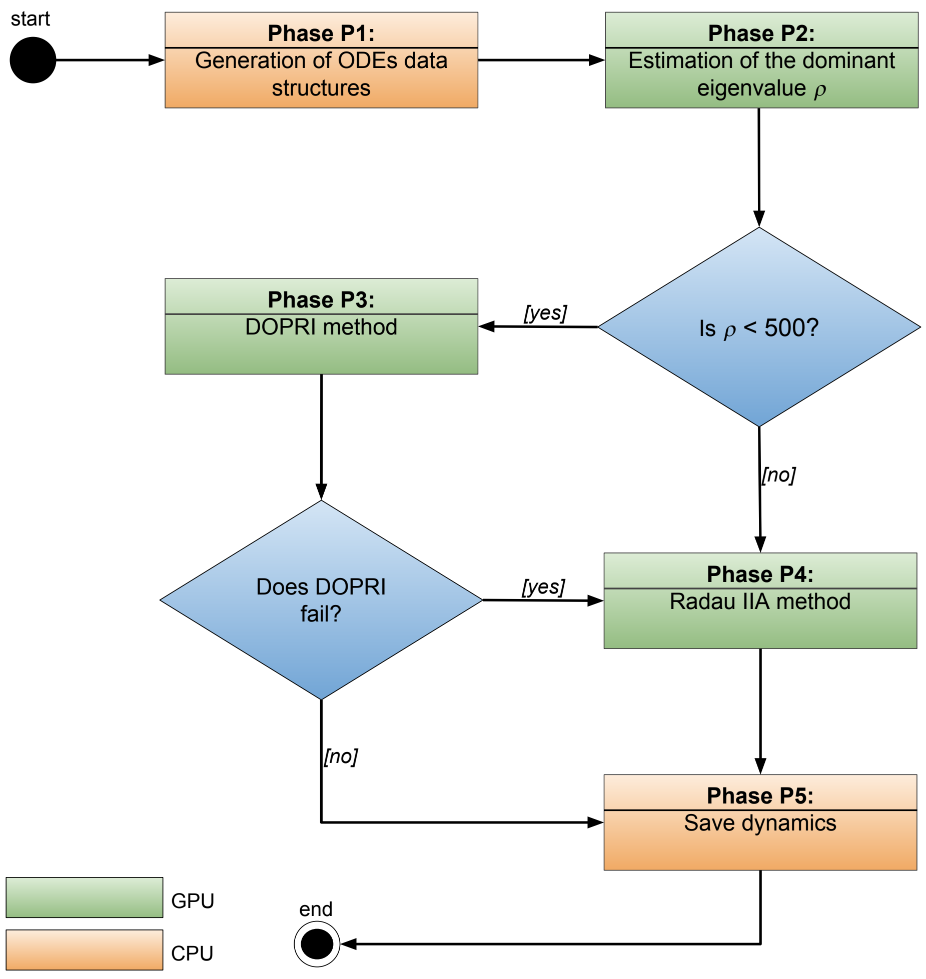


Fig. 4.14 Simplified scheme of the workflow of FiCoS. During phase P_1 all the data structures used to encode the system of ODEs are generated. In phase P_2 each thread ϕ, with $\phi = 0, \ldots \Phi - 1$, estimates the dominant eigenvalue ρ related to the simulation ϕ, using the ϕ-th parameterization and the ϕ-th set of initial concentrations. All threads whose dominant eigenvalue ρ is less than 500 are added in the set $\mathcal{L}_{DOPRI}$, which contains the threads that use DOPRI5 as integration algorithm (phase P_3). If some simulations using DOPRI5 fail, those threads are added to the set $\mathcal{L}_{RADAU}$, which contains the threads that use RADAU5 as integration algorithm (phase P_4). As soon as the phases P_3 and P_4 are completed, the output data (i.e., the concentration values of the molecular species sampled at fixed time instants) are transferred to the host that writes them to output files (phase P_5).

- kernel K_{2_a}: each thread j, with $j = 0, \ldots, N-1$, evaluates the j-th ODE using the state of the system $\mathbf{x}_d$ and the parameterization given as inputs;
- kernel K_{2_b}: it exploits N threads to update the vectors used to calculate the initial step-size of the DOPRI5 method;

- kernels K_{2_c} – K_{2_i}: each kernel uses N threads to update the vectors required by the DOPRI5 method to estimate the next step-size, using the Butcher tableau shown in Table 1.5;
- kernels K_{2_j} and K_{2_k}: in each kernel, N threads are used to update the vectors involved in the spline approximation of the ODEs.

It is worth noting that each thread $d \in \mathcal{L}_{DOPRI}$ exploits the dynamic parallelism to call the kernels K_{2_a} – K_{2_k}, launching a novel grid of threads. Since data transfers between the CPU and the GPU are very time consuming, all the temporary results computed by FiCoS during this phase are stored on the global memory of the GPU.

Phase $\mathbf{P_4}$. This phase implements the RADAU5 method [177, 178], an implicit Runge-Kutta integration algorithm of order 5 with variable step-size. It is executed after the phase P_3 since some threads running that phase can fail. The RADAU5 method is based on the following 22 kernels:

- kernel K_3: it is the main kernel implementing the RADAU5 method. It is executed by each thread $r \in \mathcal{L}_{RADAU}$, which uses its own parameterization as well as its own set of initial concentrations;
- kernel K_{1_a}: each thread j, with $j = 0, \dots, N-1$, calculates the values of the j-th row of $\mathbf{J}$ using the state of the system $\mathbf{x}_r$ and the parameterization given as inputs;
- kernel K_{2_a}: each thread j, with $j = 0, \dots, N-1$, evaluates the j-th ODE using the state of the system $\mathbf{x}_r$ and the parameterization given as inputs;
- kernel K_{3_a}: it uses N threads to reinitialize the vectors involved in the Newton-Raphson method [35], which is exploited to solve the non-linear systems (see Section 1.2.1);
- kernels K_{3_b} and K_{3_c}: each kernel takes advantage of N threads to update the matrices—which are linearized—used during the linear system resolutions required by the RADAU5 method;
- kernels K_{3_d} and K_{3_l}: in each kernel, N threads are used to update the vectors used before and after the linear system resolutions required by RADAU5 method, relying on the Butcher tableau given in Table 1.6;
- kernels K_{3_m} and K_{3_o}: these 3 kernels are used during the error estimation phase in which the new step-size is computed. Also in this case, each kernel takes advantage of

N threads to update the vectors required to calculate the new step-size based on the estimated error;

- kernels $K3_p$ and $K3_s$: in each kernel, N threads are used to update the vectors involved in the spline approximation of the ODEs.

We highlight that both the matrix decompositions and the linear system resolutions were implemented exploiting the cuBLAS library [315]. Note that each thread $r \in \mathcal{L}_{RADAU}$ takes advantage of the dynamic parallelism to call the kernels $K1_a$, $K2_a$, $K3_a - K3_s$, launching novel grids of threads. Since data transfers between the CPU and the GPU are very time consuming, all the temporary results computed by FiCoS during this phase are stored on the global memory of the GPU.

Phase $\mathbf{P_5}$. During this phase, for each simulation the dynamics of the species to be sampled are written in output files. These output data (i.e., the concentration values of molecular species sampled at fixed time instants) are transferred to the host as soon as the phases P_3 and P_4 are completed.

4.3.2 Results

In this section the computational performance of FiCoS are compared against LSODA, VODE, cupSODA, and LASSIE. We leveraged the LSODA and VODE methods provided by the SciPy scientific library (version 0.18.1) [219]; we used `Python` (version 2.7.12) and the NumPy library (version 1.11.2). All tests that follow were performed on a workstation equipped with an Intel Core i7-2600 CPU (clock 3.4 GHz) and 8 GB of RAM, running on Ubuntu 16.04 LTS. The GPU used in the tests was an Nvidia GeForce GTX Titan X (3072 cores, clock 1.075 GHz, RAM 12 GB), CUDA toolkit version 8 (driver 387.26). All the simulations were executed by using the following settings for FiCoS, LSODA, VODE, and cupSODA: (*i*) absolute tolerance $\varepsilon_a = 10^{-12}$; (*ii*) relative tolerance $\varepsilon_r = 10^{-6}$; (*iii*) maximum number of allowed steps equal to 10^4. These values correspond to the most used values in literature and by the main computational tools (e.g., COPASI [202]). LASSIE was run using the default settings. We highlight that in every simulation we stored the dynamics of all the species appearing in the tested RBMs.

The performance of FiCoS as a fine- and coarse-grained simulator was evaluated using two sets of synthetic models of increasing size, generated by SMGen (Section 4.1) in order to satisfy the following characteristics:

- a log-uniform distribution in the interval $[10^{-4}, 1)$ was applied to sample the initial concentrations of the molecular species;

- a log-uniform distribution in the interval $[10^{-6}, 10]$ was used to sample the values of the kinetic constants;
- the stoichiometric matrix **A** was generated allowing only zero, first and second-order reactions (i.e., $\max_{\text{ord}} = 2$), meaning that in each reaction at most two reactant molecules of the same or different species can appear;
- the stoichiometric matrix **B** is created allowing at most 2 product molecules for each reaction (i.e., $\max_{\text{prod}} = 2$).

A log-uniform distribution (i.e., a uniform distribution in the logarithmic space) was used to sample both the initial conditions and the kinetic constants to capture the typical dispersion of both the concentrations and kinetic parameters, which generally span over multiple orders of magnitude [485, 73].

For each synthetically generated RBM, the initial value of the kinetic parameters was perturbed to generate different parameterizations (up to 2048). For each kinetic parameter k_i, with $i = 1, \ldots, M$, we applied the following perturbation:

$$k_i = \exp(\ln(k_i - 0.25 \cdot k_i) + (\ln(k_i + 0.25 \cdot k_i) - \ln(k_i - 0.25 \cdot k_i)) \cdot rnd), \tag{4.2}$$

where rnd is a random number sampled with the uniform distribution in $[0, 1)$.

In addition to the synthetic RBMs, FiCoS was applied to carry out an in-depth investigation of the Autophagy/Translation Switch Based on Mutual Inhibition of MTORC1 and ULK1 model [422], as an interesting case study of a real large-scale biological system.

4.3.3 Computational performance

By using the two sets of synthetic RBMs, we performed a thorough analysis to understand which integration method is the most suitable under different conditions (i.e., number of species and number of reactions). We investigated how the relationship between the number of reactions and the number of species affects the overall performances of the GPU-powered simulators. As already stated for LASSIE in Section 4.2, since each ODE corresponds to a different species, the higher the number of the species the higher the parallelization that can be achieved by exploiting the fine-grained strategy. On the contrary, the number of reactions is roughly related to the length of each ODE, thus increasing the number of operations that must be performed by each thread. Since the clock frequency of the GPUs cores is lower with respect to the CPUs ones, the required time to perform a single instruction on a GPU is higher. Thus, the performance that can be achieved by means of GPU-powered simulators decreases when the number of reactions is much larger than the number of species.

Symmetric models. Figure 4.15 depicts the map representing the comparison of the achieved results considering the symmetric RBMs. According to our results, FiCoS provides a relevant reduction of the simulation time, achieving a 360× speed-up against LSODA (Figure 4.16a), and a 487× speed-up against VODE (Figure 4.17a) in the case of 2048 parallel simulations of the 800×800 model. This is the highest speed-up achieved with respect to VODE, while against LSODA the maximum speed-up (366×) was reached considering 1024 parallel simulations of the 800×800 model. For bigger models, the speed-up could be even higher with respect to CPU-bound execution; however, we did not perform tests using bigger models due to the excessive memory requirements afflicting both LSODA and VODE implementations.

Since FiCoS was designed to perform both fine- and coarse-grained parallelization strategies, the intra-GPU communication overhead—due to the fine-grain kernels—causes the simulation to become inefficient in the case of small size models. Moreover, as expected, the performance of FiCoS can be affected when only a few parallel simulations are executed. For instance, in the case of a single simulation of a model smaller than or equal to 256×256, FiCoS is slower than both LSODA (Figure 4.16a) and VODE (Figure 4.17a). Thus, whilst for a single simulation of a small/medium model a CPU-bound simulation should be preferred, in any other case FiCoS is highly beneficial. Moreover, the coarse-grained performance can be affected by the saturation of the computing resources due to excessive parallelization. The computing resources required by the fine-grained kernels, which are exploited to parallelize the resolution of the system of ODEs, increase along with the model size. As a matter of fact, the blocks of threads generated using the dynamic parallelism saturate the GPU resources, thus decreasing the speed-up. In our tests, this phenomenon occurred for models larger than 512 species and reactions, in the case of 2048 parallel simulations.

We now restrict the analyses to the actual integration time. Notice that these tests exclude from the analyses the portion of the running time due to reading/writing operations of the input/output files. According to our tests, the speed-up achieved by the integration method implemented in FiCoS is smaller than the overall acceleration of the simulation time in the case of LSODA. Specifically, the highest speed-up is 79× with respect to LSODA integration, and 855× with respect to VODE (see Figs. 4.16b and 4.17b, respectively). The patterns in the results, however, are consistent with the previous findings: FiCoS becomes less effective in scarcity of resources, e.g., when more than 2000 parallel simulations are executed. These results show that the integration method used in FiCoS are very effective and efficient.

We evaluated the performance of FiCoS also against LASSIE: as shown in Figure 4.18a, LASSIE is faster than FiCoS in the case of a single simulation (except when 512 species and reactions are considered). Regarding the integration time, FiCoS always outperformed

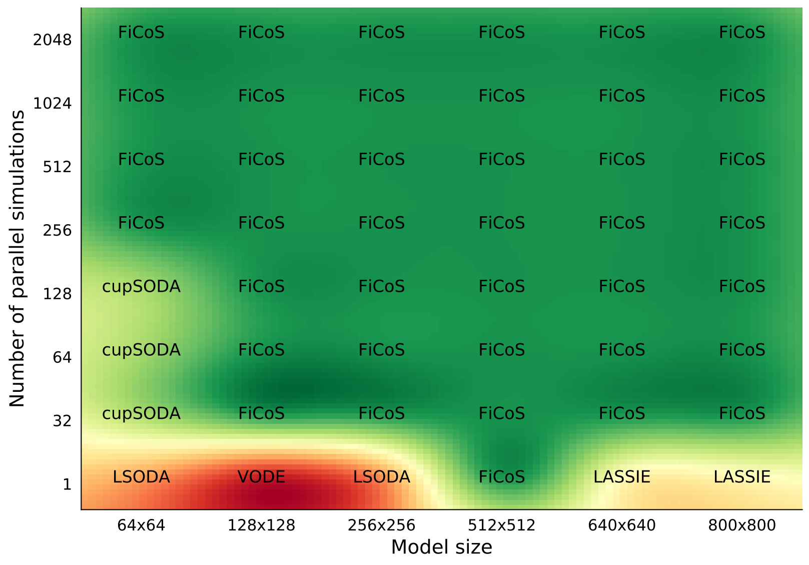


Fig. 4.15 Comparison map representing the best method in terms of simulation time in the case of symmetric models. The simulation time was analyzed for the increasing size of the models (horizontal axis) and number of parallel simulations (vertical axis).

LASSIE achieving a maximum speed-up equal to 5.57× (Fig. 4.18a). Considering more than a simulation, FiCoS resulted always faster than LASSIE, with the highest speed-up achieved in the case of 2048 simulations of the model characterized by 64 species and reactions. In such a case, the simulation time of FiCoS is 298× slower than that required by LASSIE (Fig. 4.18a), while the integration time is 760× slower (Fig. 4.18b).

In order to evaluate the coarse-grained performance of FiCoS, we tested it against cupSODA. Since cupSODA was designed for small models, it resulted faster than FiCoS in the case of a single simulation when less than 512 species and reactions are taken into account (see Fig. 4.19a). In the case of the smallest model (i.e., 64 species and reactions), FiCoS is more efficient only considering more than 128 simulations. In the remaining cases, FiCoS turns out to be the natural choice achieving a speed-up up to 7.35×. Regarding the integration time, FiCoS always outperformed cupSODA, reaching the maximum speed-up (17×) in the case of 128 simulations of the model characterized by 800 species and reactions (see Fig. 4.19b). Notice that, due to the required amount of memory, cupSODA failed to perform 2048 simulations of the models with 640 and 800 species and reactions.

To summarize, in the case of a single simulation, LSODA and VODE are the best choice when dealing with models characterized by a number of species and reactions lower than 512×512, which represents the break-even point among CPU- and GPU-based simulators.

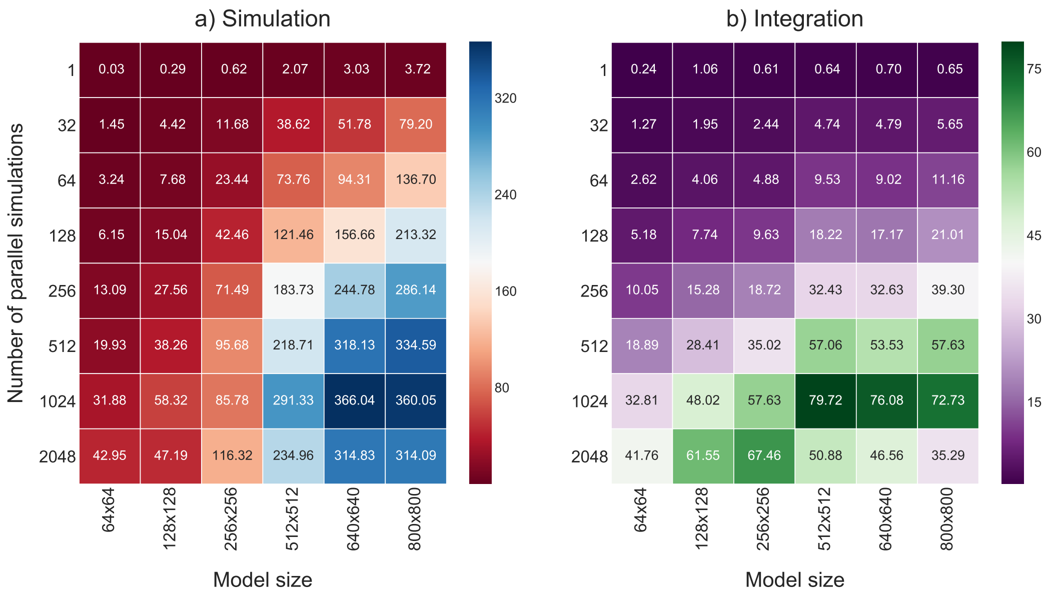


Fig. 4.16 Speed-up provided by FiCoS with respect to LSODA considering the symmetric models. The speed-up is analyzed for an increasing size of the model (horizontal axis) and number of parallel simulations (vertical axis). Panel a) shows the speed-up regarding the whole simulation, while panel b) considers only the numerical integration.

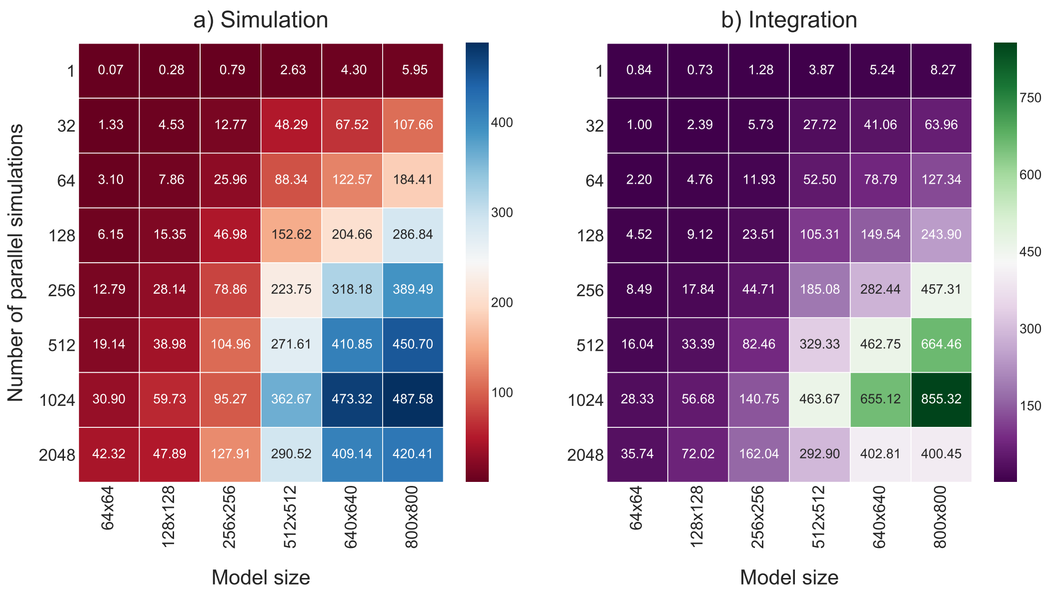


Fig. 4.17 Speed-up provided by FiCoS with respect to VODE considering the symmetric models. The speed-up is analyzed for an increasing size of the model (horizontal axis) and number of parallel simulations (vertical axis). Panel a) shows the speed-up regarding the whole simulation, while panel b) considers only the numerical integration.

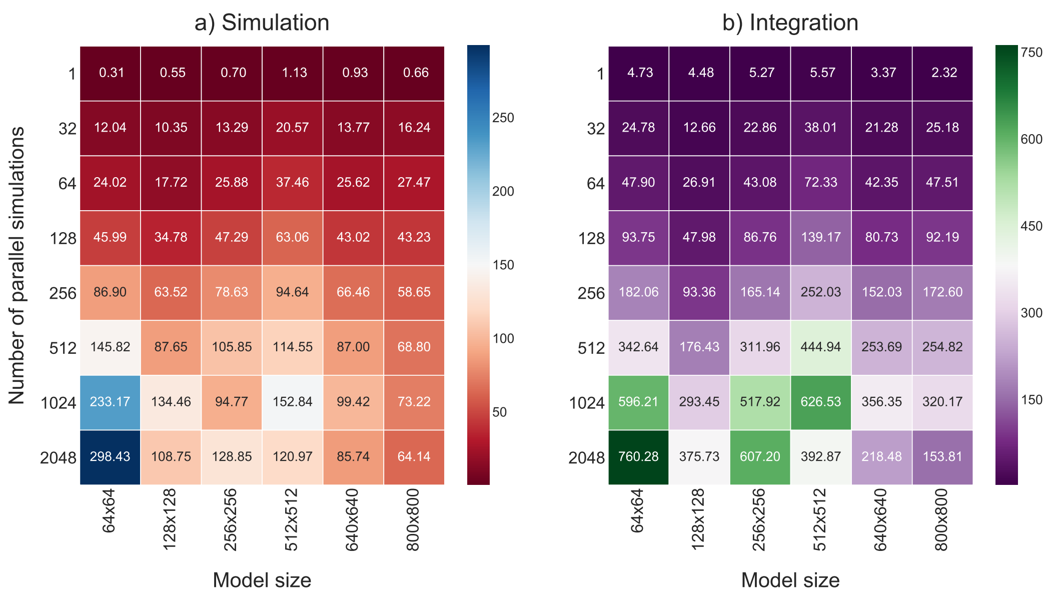


Fig. 4.18 Speed-up provided by FiCoS with respect to LASSIE considering the symmetric models. The speed-up is analyzed for an increasing size of the model (horizontal axis) and number of parallel simulations (vertical axis). Panel a) shows the speed-up regarding the whole simulation, while panel b) considers only the numerical integration.

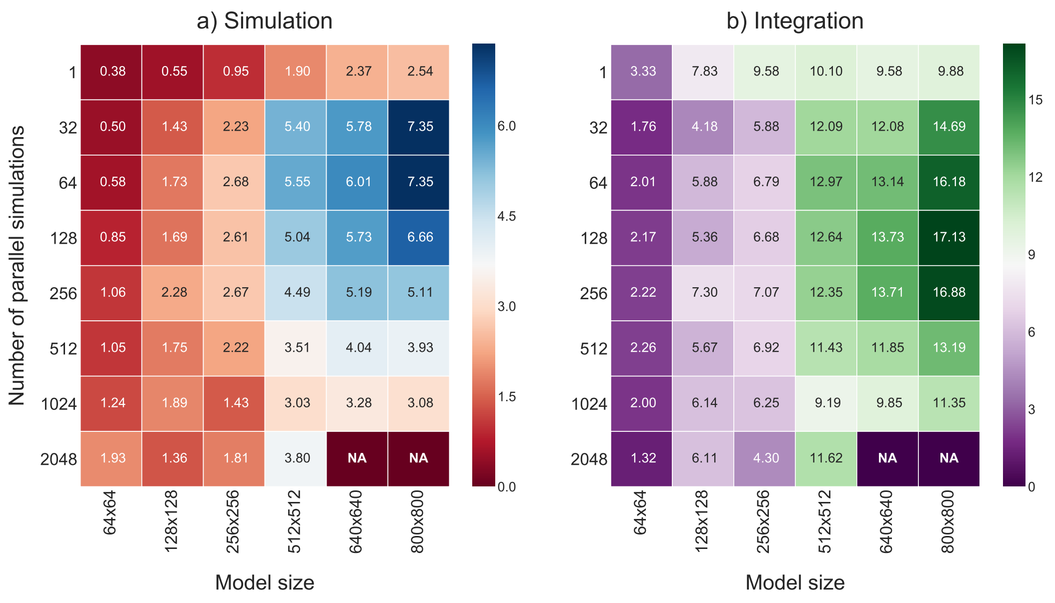
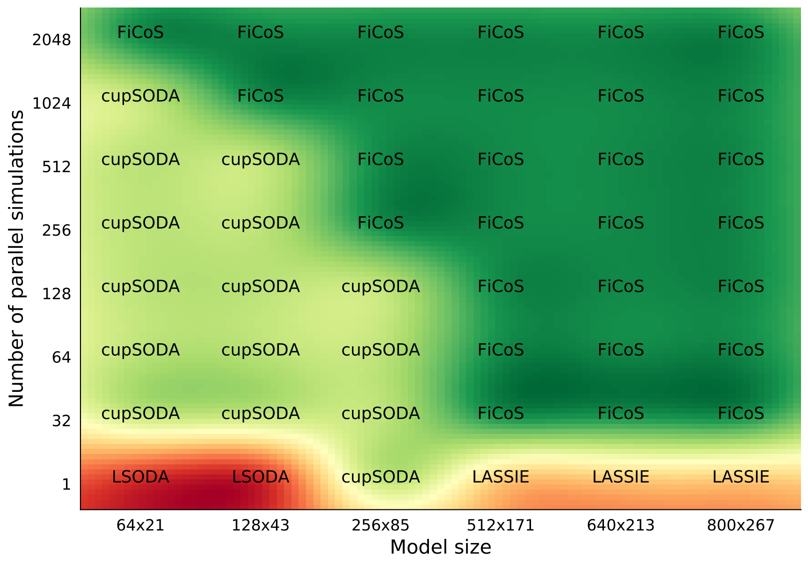


Fig. 4.19 speed-up provided by FiCoS with respect to cupSODA considering the symmetric models. The speed-up is analyzed for an increasing size of the model (horizontal axis) and number of parallel simulations (vertical axis). Panel a) shows the speed-up regarding the whole simulation, while panel b) considers only the numerical integration. The NA values indicate that cupSODA failed to perform the simulations due to the required amount of memory.



	64x21	128x43	256x85	512x171	640x213	800x267
2048	FiCoS	FiCoS	FiCoS	FiCoS	FiCoS	FiCoS
1024	cupSODA	FiCoS	FiCoS	FiCoS	FiCoS	FiCoS
512	cupSODA	cupSODA	FiCoS	FiCoS	FiCoS	FiCoS
256	cupSODA	cupSODA	FiCoS	FiCoS	FiCoS	FiCoS
128	cupSODA	cupSODA	cupSODA	FiCoS	FiCoS	FiCoS
64	cupSODA	cupSODA	cupSODA	FiCoS	FiCoS	FiCoS
32	cupSODA	cupSODA	cupSODA	FiCoS	FiCoS	FiCoS
1	LSODA	LSODA	cupSODA	LASSIE	LASSIE	LASSIE

Fig. 4.20 Comparison map representing the best method in terms of simulation time in the case of asymmetric models ($N \times M$), where N indicates the number of species and M the number of reactions, and $N > M$. The simulation time was analyzed for the increasing size of the models (horizontal axis) and number of parallel simulations (vertical axis).

When models characterized by a larger number of species and reactions are taken into account, LASSIE is the best choice thanks to its design and implementation that rely on a pure fine-grained strategy. Considering multiple simulations, cupSODA should be used with small-scale models (i.e., less than 128 species and reactions) and a number of parallel simulations less than 256. As a matter of fact, considering these dimensions cupSODA is capable of exploiting the most performing GPU memories (i.e, constant and shared memories). In the other cases, FiCoS outperforms all the simulators thanks to its mixed fine- and coarse-grained parallelization strategies that allow for distributing both the number of simulations and the calculations required to solve the system of ODEs.

Asymmetric models. In order to evaluate how the number of species and reactions affects the performance of the simulators, especially those exploiting GPUs to reduce the required simulation time, we designed specific tests. We generated two sets of asymmetric models: in the former we used a number of species that is three times higher than the number of reactions, while in the latter we flipped this condition.

We remind that since each ODE corresponds to a different species, the higher the number of the species the higher the parallelization that can be obtained by exploiting the fine-grained

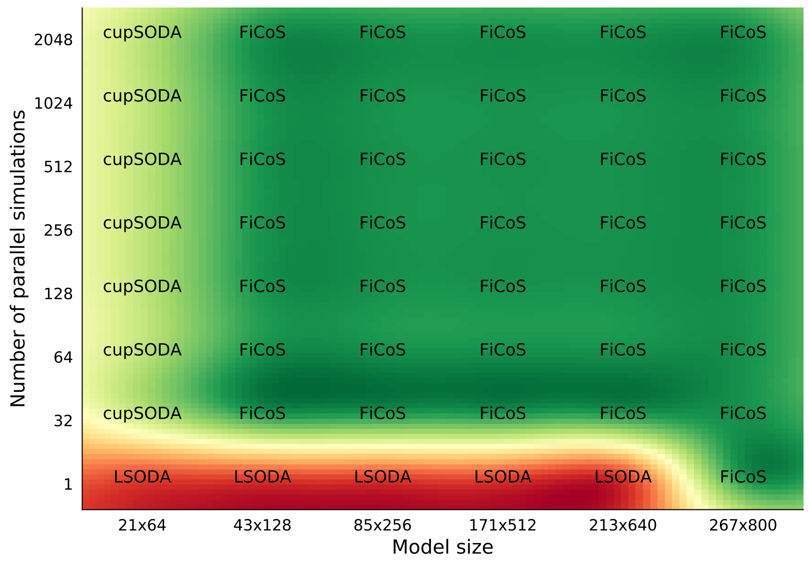


Fig. 4.21 Comparison map representing the best method in terms of simulation time in the case of asymmetric models ($N \times M$), where N indicates the number of species and M the number of reactions, and $M > N$. The simulation time was analyzed for the increasing size of the models (horizontal axis) and number of parallel simulations (vertical axis).

strategy. Conversely, since the number of reactions is roughly related to the length of each ODE, the number of operations that must be performed by each thread increases along with the number of reactions. Considering this condition as well as the clock frequency, which is slower in the case of GPUs cores with respect to the CPUs ones, the simulation time required to perform a single instruction on GPU is higher. In such a case, the performance that can be reached by means of GPU-powered simulators should decrease when the number of reactions is much larger than the number of species.

Figures 4.20 and 4.21 show the results considering the asymmetric models. As in the case of symmetric models, for each synthetic model we performed an increasing number of parallel simulations (up to 2048) and calculated both the integration time and the overall simulation time. Figures 4.22, 4.23, 4.24 and 4.25 show the speed-ups achieved by FiCoS against LSODA, VODE, LASSIE and cupSODA, respectively, considering a number of species higher than the number of reactions. Conversely, Figures 4.26, 4.27, 4.28 and 4.29 show the speed-ups achieved in the case of a higher number of reactions. Taking into account the simulation time, FiCoS achieved a speed-up up to 413$\times$ against LSODA, and up to 508$\times$ with respect to VODE (see Figures 4.22a, 4.23a, respectively). These values are higher with respect to those obtained considering the symmetric models (Figures 4.16a and 4.17a).

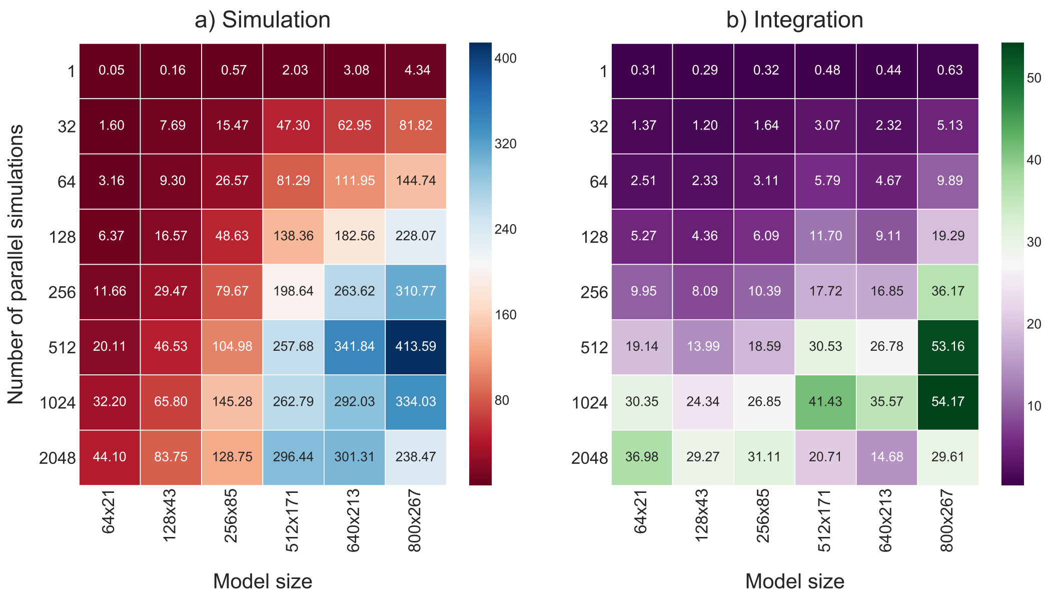


Fig. 4.22 Speed-up provided by FiCoS with respect to LSODA considering the asymmetric models in which the number of species is three times higher than the number of reactions. The speed-up is analyzed for an increasing size of the model (horizontal axis) and number of parallel simulations (vertical axis). Panel a) shows the speed-up regarding the whole simulation, while panel b) considers only the numerical integration.

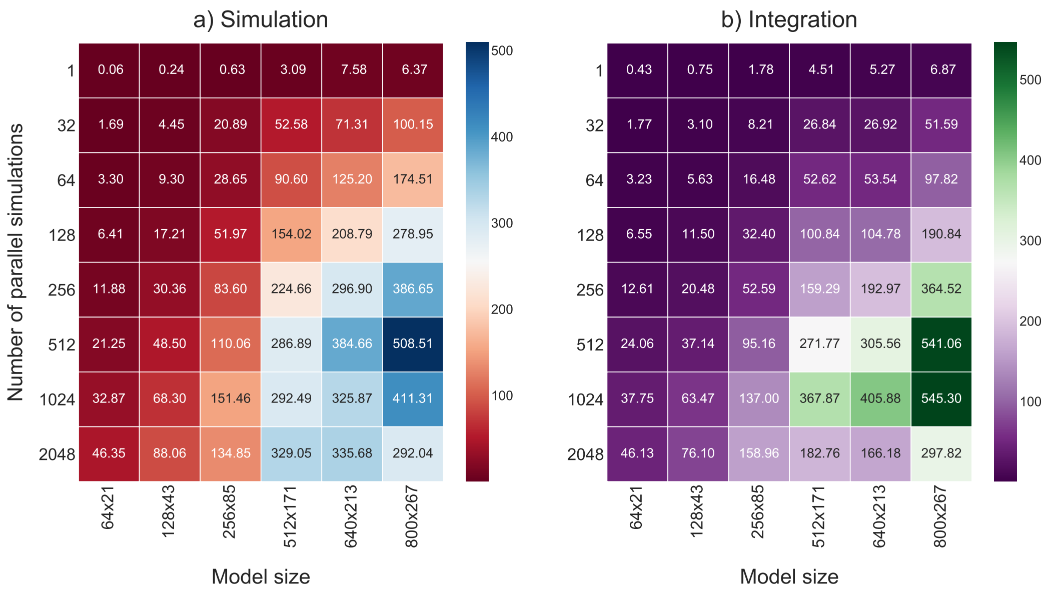


Fig. 4.23 Speed-up provided by FiCoS with respect to VODE considering the asymmetric models in which the number of species is three times higher than the number of reactions. The speed-up is analyzed for an increasing size of the model (horizontal axis) and number of parallel simulations (vertical axis). Panel a) shows the speed-up regarding the whole simulation, while panel b) considers only the numerical integration.

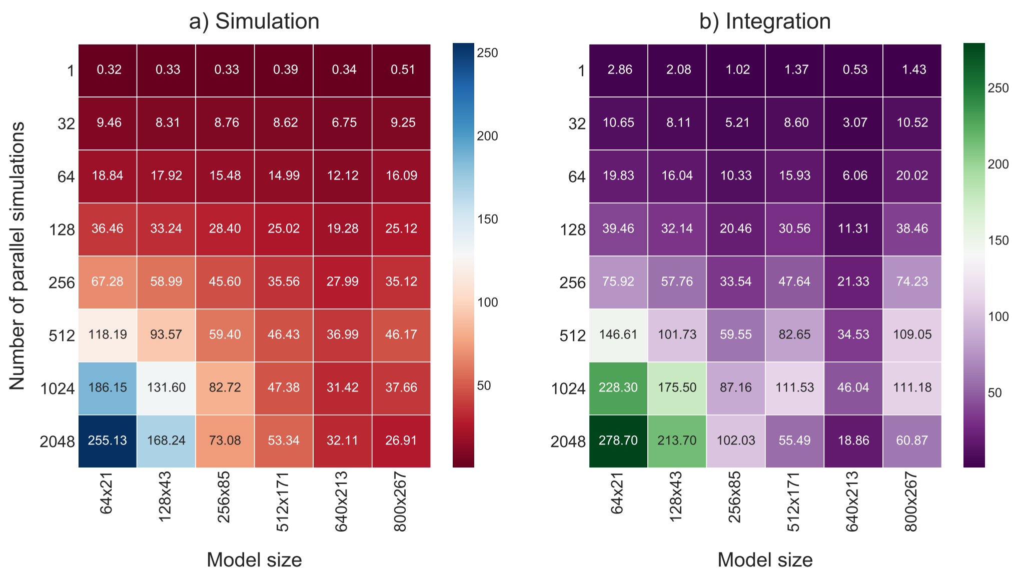


Fig. 4.24 Speed-up provided by FiCoS with respect to LASSIE considering the asymmetric models in which the number of species is three times higher than the number of reactions. The speed-up is analyzed for an increasing size of the model (horizontal axis) and number of parallel simulations (vertical axis). Panel a) shows the speed-up regarding the whole simulation, while panel b) considers only the numerical integration.

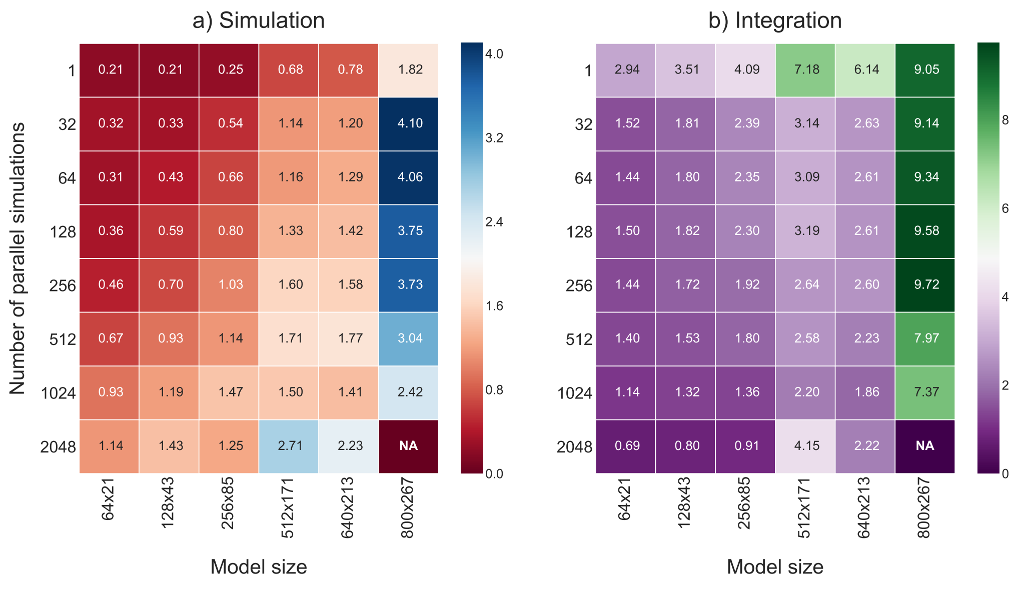


Fig. 4.25 Speed-up provided by FiCoS with respect to cupSODA considering the asymmetric models in which the number of species is three times higher than the number of reactions. The speed-up is analyzed for an increasing size of the model (horizontal axis) and number of parallel simulations (vertical axis). Panel a) shows the speed-up regarding the whole simulation, while panel b) considers only the numerical integration. The NA values indicate that cupSODA failed to perform the simulations due to the required amount of memory.

Conversely, regarding the integration time, the speed-ups achieved by FiCoS are smaller compared to those obtained by simulating the symmetric models.

Since the fine-grained strategy is exploited to parallelize the number of ODEs, whose length is shorter compared to those of symmetric models, the speed-up achieved with respect to LASSIE is smaller. As a matter of fact, LASSIE always outperformed FiCoS when a single simulation is performed. Considering the simulation time, the maximum speed-up decreased from 298× to 255×, while in the case of the integration time it decreased from 760× (simulation time) to 278× (see Figures 4.24 and 4.18). Finally, comparing FiCoS and cupSODA we can note that the maximum speed-up is approximately half the one achieved considering the symmetric models (see Figures 4.25 and 4.19).

Considering the set of asymmetric models characterized by a higher number of reactions, FiCoS remains faster than both LSODA and VODE, but the speed-ups, regarding the simulation time, are smaller than those achieved in the case of both symmetric and asymmetric (with more species than reactions) models. As shown in Figures 4.26a and 4.27a, the best results are equal to 252× (LSODA) and 277× (VODE). Taking into account the integration time, FiCoS achieved a maximum speed-up equal to 147× with respect to LSODA (Figure 4.26b), which represents the best result. In the case of the comparison against VODE (Figure 4.27b), the maximum speed-up is 234×, which is the smallest one considering this method.

LASSIE resulted always slower than FiCoS, which was capable of achieving the maximum speed-ups (1100× considering the simulation time and 3666× in the case of the integration time, as depicted in Figure 4.28). This result was possible since the number of species is smaller compared to that of symmetric models and asymmetric models with a higher number of species than reactions. In such a case, the pure fine-grained parallelization strategy resulted inefficient.

Finally, the comparison between FiCoS and cupSODA indicates that FiCoS achieved the best result in this case. As shown in Figure 4.29, regarding the simulation time it obtained a maximum speed-up equal to 27×, while regarding the integration time the maximum speed-up increased up to 60×.

To summarize, only considering a small number of species (less than 256) and a single simulation, LSODA can be effectively applied. As in the case of symmetric models, when a single simulation is required, LASSIE is the fastest simulator if the number of species becomes greater than 256. cupSODA is the best simulator to perform a number of simulations less than 2048 when the number of species is not greater than 512. Otherwise, FiCoS should be exploited to reduce the required simulation time. Taking into account the models in which the number of reactions is considerably greater than the number of species, LSODA is the natural choice to perform a single simulation for models up to 213 species.

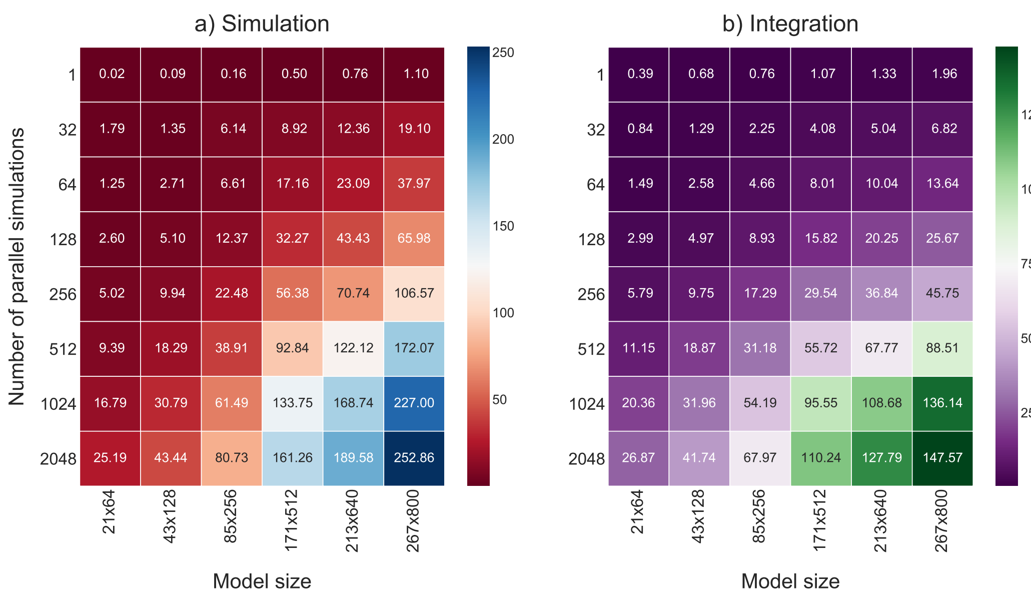


Fig. 4.26 Speed-up provided by FiCoS with respect to LSODA considering the asymmetric models in which the number of reactions is three times higher than the number of species. The speed-up is analyzed for an increasing size of the model (horizontal axis) and number of parallel simulations (vertical axis). Panel a) shows the speed-up regarding the whole simulation, while panel b) considers only the numerical integration.

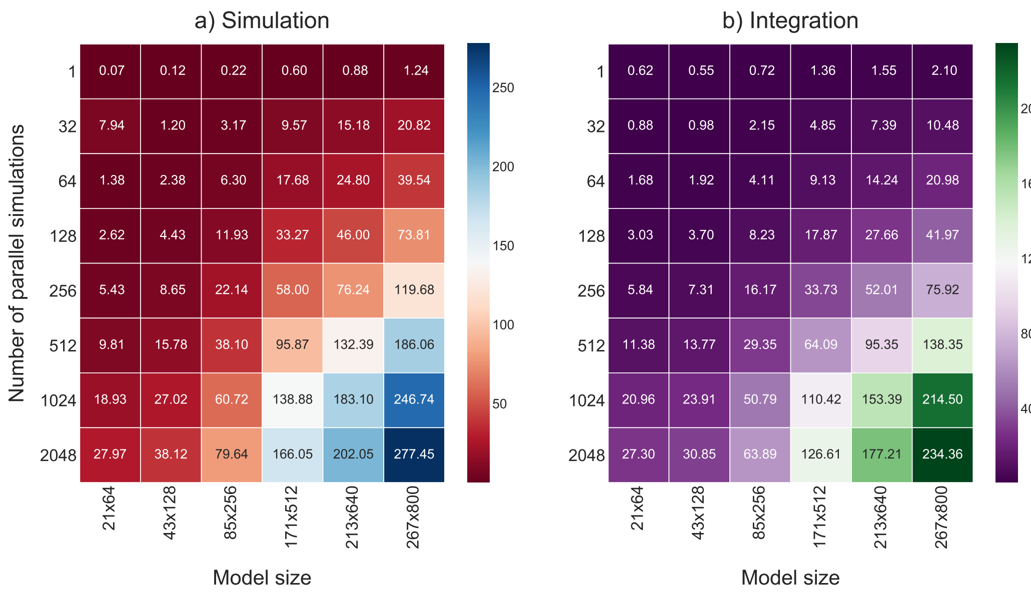


Fig. 4.27 Speed-up provided by FiCoS with respect to VODE considering the asymmetric models in which the number of reactions is three times higher than the number of species. The speed-up is analyzed for an increasing size of the model (horizontal axis) and number of parallel simulations (vertical axis). Panel a) shows the speed-up regarding the whole simulation, while panel b) considers only the numerical integration.

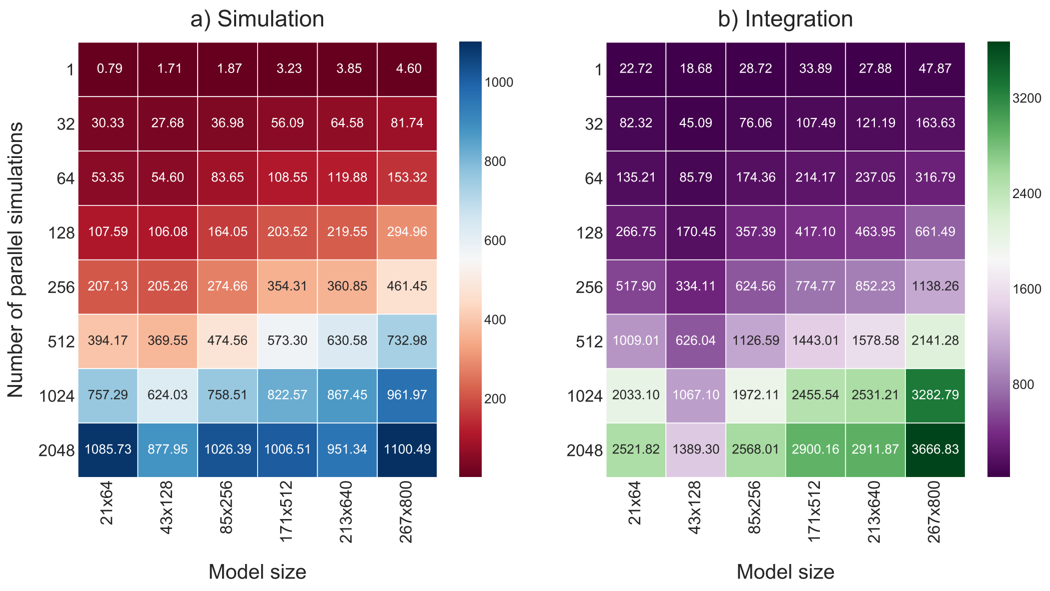


Fig. 4.28 Speed-up provided by FiCoS with respect to LASSIE considering the asymmetric models in which the number of reactions is three times higher than the number of species. The speed-up is analyzed for an increasing size of the model (horizontal axis) and number of parallel simulations (vertical axis). Panel a) shows the speed-up regarding the whole simulation, while panel b) considers only the numerical integration.

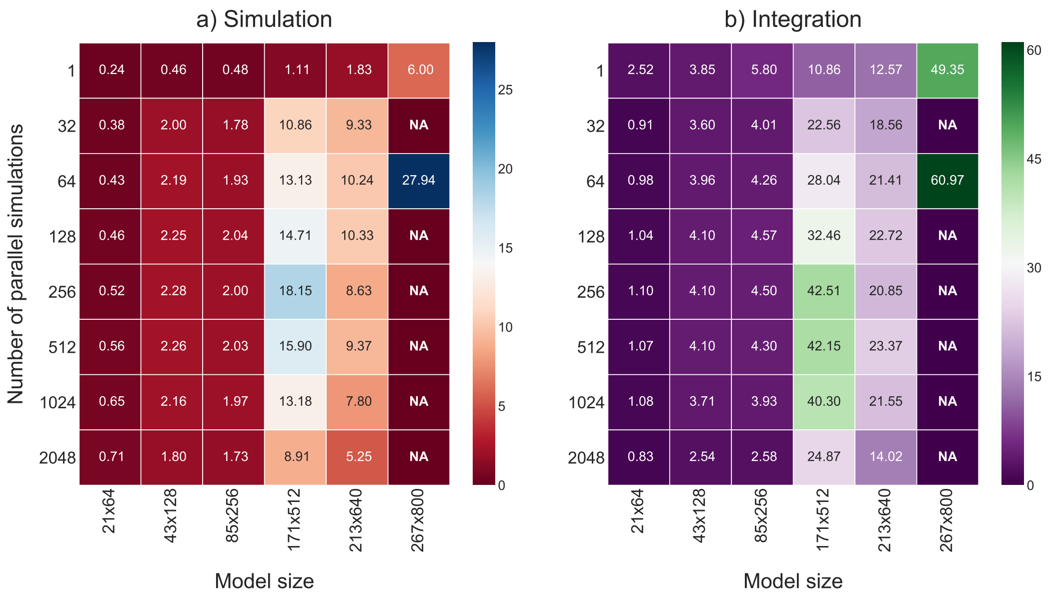


Fig. 4.29 Speed-up provided by FiCoS with respect to cupSODA considering the asymmetric models in which the number of reactions is three times higher than the number of species. The speed-up is analyzed for an increasing size of the model (horizontal axis) and number of parallel simulations (vertical axis). Panel a) shows the speed-up regarding the whole simulation, while panel b) considers only the numerical integration. The NA values indicate that cupSODA failed to perform the simulations due to the required amount of memory.

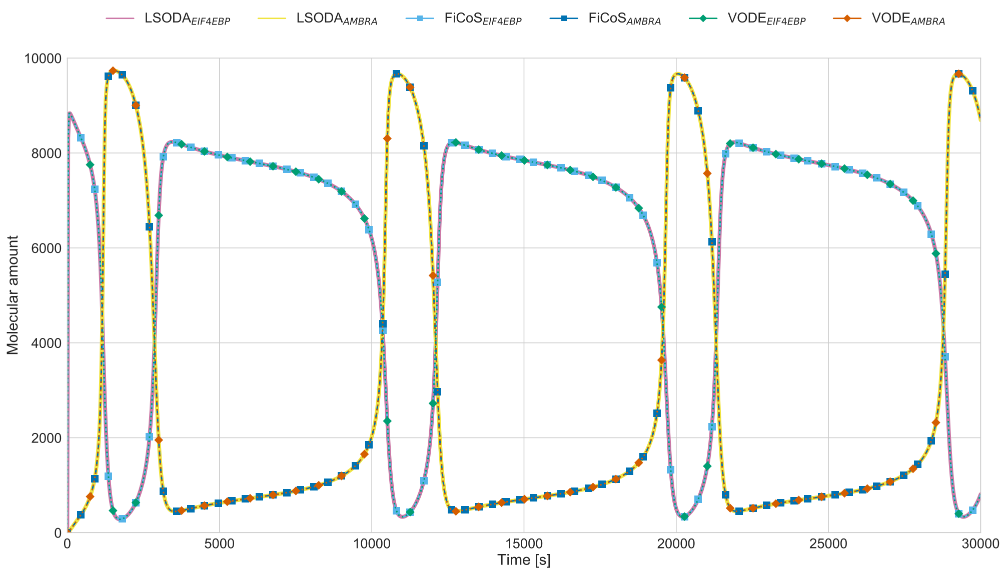


Fig. 4.30 Comparison of the dynamics of the molecular species EIF4EBP and AMBRA, obtained by running FiCoS, LSODA and VODE, by using $\varepsilon_a = 10^{-12}$ and $\varepsilon_r = 10^{-6}$.

This result confirms that the number of reactions has a deep impact on the performance that can be achieved by exploiting the GPUs. FiCoS outperforms all the other simulators when larger models are considered as well as in the case of multiple simulations. Finally, it is worth noting that in the case of small-scale models, cupSODA remains the best tool to perform a massive number of simulations.

This analysis shows the importance of GPU-powered deterministic simulators, especially when a massive number of simulations must be performed or large-scale models are taken into account. These results show that the pure fine- and coarse-grained strategies taken alone are inefficient in some conditions, while a mixed strategy is capable of obtaining great results in all the tested conditions.

4.3.4 Simulation accuracy

In order to assess the accuracy of FiCoS simulations, we compared the dynamics of the molecular species EIF4EBP and AMBRA of the model of the Autophagy/Translation switch based on the mutual inhibition of MTORC1 and ULK1 [422] (see Section 4.3.5 for more details). The dynamics were obtained by running LSODA, VODE and FiCoS. We set the initial value of AMPK to 90000 (molecules/cell), as suggested in [422], to induce an oscillatory behavior that makes the system of ODEs stiff. Figure 4.30 shows that the dynamics obtained by FiCoS perfectly overlap those reproduced by LSODA and VODE, confirming FiCoS accuracy.

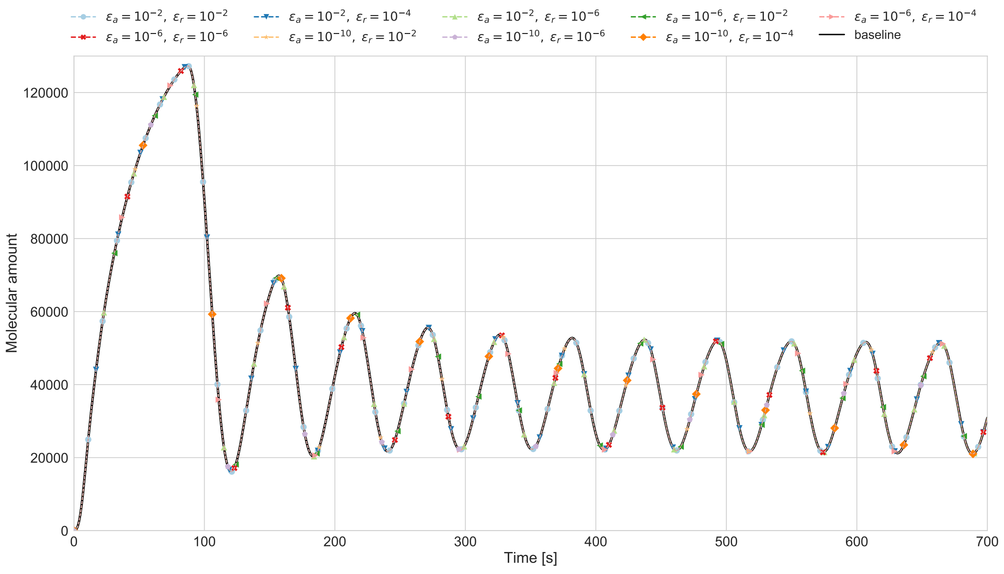


Fig. 4.31 Comparison of the dynamics of the molecular species cAMP of the model of the Ras/cAMP/PKA signaling pathway in yeast, obtained by FiCoS varying $\varepsilon_a \in \{10^{-2}, 10^{-6}, 10^{-10}\}$ and $\varepsilon_r \in \{10^{-2}, 10^{-4}, 10^{-6}\}$.

In order to assess the robustness of FiCoS with respect to its settings (i.e., absolute (ε_a) and relative (ε_r) tolerances), we designed an *ad hoc* batch of tests in which we changed one parameter at a time. The same batch was also executed by running both LSODA and VODE. We evaluated the effect of these settings on both the quality of the output dynamics and the simulation time. For the execution of these tests we considered the model of the Ras/cAMP/PKA signal transduction pathway in yeast [44, 75, 339] (see Appendix A.5 for further details). The accuracy of the simulations performed by LSODA, VODE and FiCoS were assessed by comparing the dynamics of the molecular species cAMP, simulated by using different settings. The baseline shown in Figures 4.31, 4.32 and 4.33 was obtained by running LSODA with $\varepsilon_a = 10^{-12}$, $\varepsilon_r = 10^{-6}$, and a maximum number of allowed steps equal to 10^4. The various simulations were performed by using the following settings:

- absolute tolerance $\varepsilon_a \in \{10^{-2}, 10^{-6}, 10^{-10}\}$;
- relative tolerance $\varepsilon_r \in \{10^{-2}, 10^{-4}, 10^{-6}\}$;
- maximum number of internal steps equal to 10^4.

Figure 4.31 depicts the output dynamics obtained by FiCoS varying the settings. Since all the dynamics are perfectly overlapped, these tolerances have no relevant effect on the quality of the output dynamics. Similarly to FiCoS, VODE generates simulated dynamics that overlap the baseline (see Figure 4.32). Conversely, the tolerance values have a high

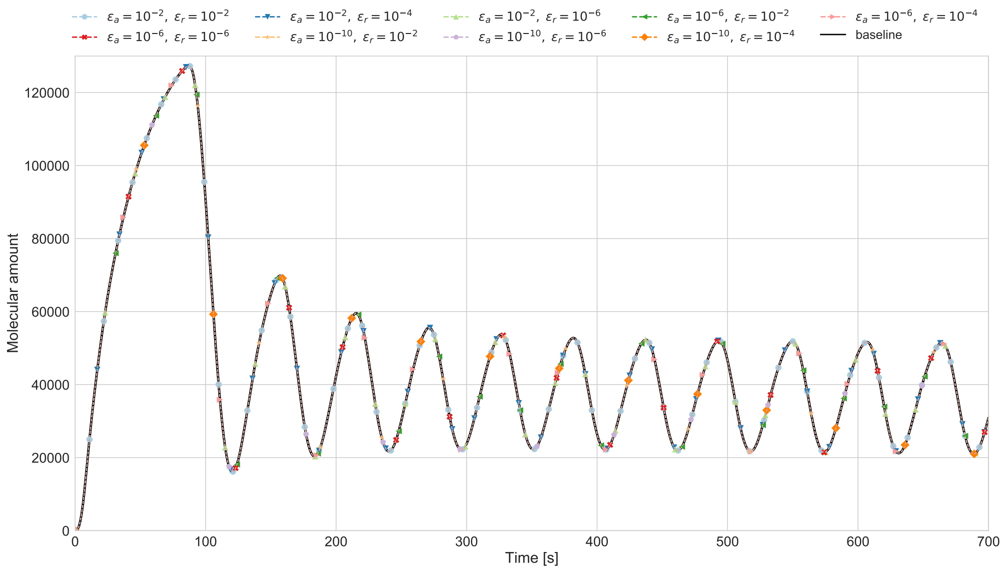


Fig. 4.32 Comparison of the dynamics of the molecular species cAMP of the model of the Ras/cAMP/PKA signaling pathway in yeast, obtained by VODE varying $\varepsilon_a \in \{10^{-2}, 10^{-6}, 10^{-10}\}$ and $\varepsilon_r \in \{10^{-2}, 10^{-4}, 10^{-6}\}$.

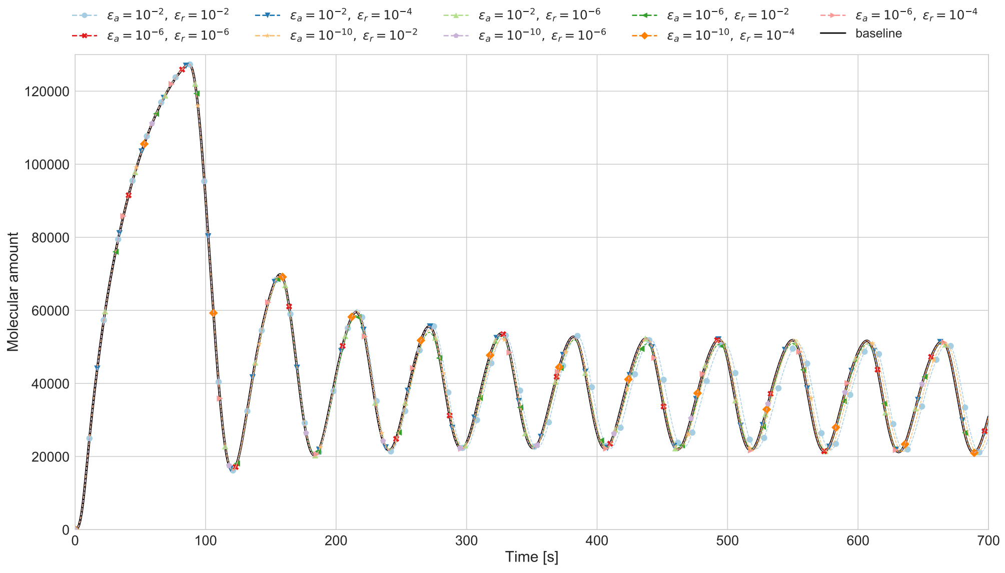


Fig. 4.33 Comparison of the dynamics of the molecular species cAMP of the model of the Ras/cAMP/PKA signaling pathway in yeast, obtained by LSODA varying $\varepsilon_a \in \{10^{-2}, 10^{-6}, 10^{-10}\}$ and $\varepsilon_r \in \{10^{-2}, 10^{-4}, 10^{-6}\}$.

impact on the dynamics produced by LSODA. As a matter of fact, when $\varepsilon_r = 10^{-2}$ and $\varepsilon_a = 10^{-2}$ or 10^{-6} are taken into account, then the output dynamics do not perfectly overlap the baseline (Figure 4.33). This result is probably related to the automatic switching between

Table 4.5 Simulation time required by LSODA, VODE and the CPU-based version of FiCoS to solve the systems of ODEs related to the Ras/cAMP/PKA signaling pathway in yeast, by varying both the absolute (ε_a) and relative (ε_r) tolerances.

Settings		Simulation time [s]		
ε_a	ε_r	LSODA	VODE	FiCoS
10^{-2}	10^{-2}	0.255	2.516	0.030
10^{-2}	10^{-4}	0.306	2.538	0.041
10^{-2}	10^{-6}	0.381	2.419	0.030
10^{-6}	10^{-2}	0.339	2.508	0.043
10^{-6}	10^{-4}	0.407	2.465	0.059
10^{-6}	10^{-6}	0.579	2.419	0.081
10^{-10}	10^{-2}	0.324	2.533	0.046
10^{-10}	10^{-4}	0.373	2.481	0.034
10^{-10}	10^{-6}	0.601	2.501	0.052

Table 4.6 Simulation time required by LSODA, VODE and the CPU-based version of FiCoS to solve the systems of ODEs related to the Autophagy/Translation switch based on the mutual inhibition of MTORC1 and ULK1 model, by varying both the absolute (ε_a) and relative (ε_r) tolerances.

Settings		Simulation time [s]		
ε_a	ε_r	LSODA	VODE	FiCoS
10^{-2}	10^{-2}	12.599	25.922	2.398
10^{-2}	10^{-4}	16.348	27.101	2.791
10^{-2}	10^{-6}	19.722	29.449	3.311
10^{-6}	10^{-2}	13.261	31.077	2.661
10^{-6}	10^{-4}	19.433	32.822	3.717
10^{-6}	10^{-6}	33.544	43.372	5.655
10^{-10}	10^{-2}	13.747	32.503	2.774
10^{-10}	10^{-4}	21.219	33.860	4.329
10^{-10}	10^{-6}	37.02	56.687	6.952

the explicit and implicit families of integration methods. Since the explicit methods are not stiffly accurate they are not able to properly integrate a stiff system, thus reducing the accuracy of the outcome.

Considering the achieved results, which provided relevant speed-ups, as well as the accuracy of the employed integration methods, we developed a CPU-based version of FiCoS to perform a single simulation. Table 4.5 summarizes the simulation time required by LSODA, VODE and the CPU-based version of FiCoS to solve the system of ODEs related to the Ras/cAMP/PKA signaling pathway in yeast, for each tested combination of ε_a and

ε_r. Every combination does not have a deep impact on VODE performance, while in the case of the CPU-based version of FiCoS and LSODA some combinations seem to double or triple the simulaion time. Since the simulations of this model are performed very quickly, we defined another batch of tests in which we varied the tolerances to analyze the simulation time required to simulate the Autophagy/Translation model (Section 4.3.5), which is characterized by by 173 molecular species and 6581 reactions. Table 4.6 shows that the simulation time of VODE doubles from the combination $\varepsilon_a = \varepsilon_r = 10^{-2}$ to the combination $\varepsilon_a = 10^{-10}$ and $\varepsilon_a = 10^{-6}$, while the simulation time of LSODA and the CPU-based version of FiCoS triples. The relative tolerance has a higher impact on the simulation time with respect to the absolute tolerance. Considering FiCoS, setting $\varepsilon_a = 10^{-10}$ the simulation time triples when using $\varepsilon_r = 10^{-6}$ with respect to $\varepsilon_r = 10^{-2}$.

Finally, considering the Autophagy/Translation model, we can observe that the CPU-based version of FiCoS is $\sim 6\times$ faster than LSODA, and $\sim 12\times$ faster than VODE. Taking into account the Ras/cAMP/PKA signal transduction pathway in yeast, the CPU-based version of FiCoS is capable of outperforming both LSODA and VODE, achieving a $\sim 12\times$ speed-up against LSODA, and a $\sim 84\times$ speed-up against VODE.

4.3.5 Autophagy/Translation model

We tested the performance of FiCoS on a model of the Autophagy/Translation switch based on the mutual inhibition of MTORC1 and ULK1 [422]. These two proteins are part of a large regulatory network that is responsible of maintaining cellular energy and nutrients homeostasis. Autophagic processes are involved in cell survival during starvation and clearing of damaged cellular components, and thus their study plays a fundamental role in understanding aging-related and neurodegenerative diseases, as well as immunity and tumorigenesis. This model, defined to investigate the complex interplay between autophagy and protein synthesis, was defined using BioNetGen rule-based modeling language (BNGL) [186] and is characterized by 7 initial molecule types involved in 29 rules. Rule-based modeling is a practical means to succintely describe the interactions between molecules, keeping track of site-specific details (e.g., the phosphorylation states) (see Section 1.1.1). In the case of the rule-based model of the Autophagy/Translation system, the corresponding asymmetric RBM contains 173 molecular species involved in 6581 reactions[2].

FiCoS was capable of achieving relevant speed-ups with respect to LSODA and VODE for the execution of 512 simulations. For what concerns the simulation time, FiCoS is $22\times$

[2]The parameterizations of this model were obtained using the same procedure exploited for the generation of the synthetic models.

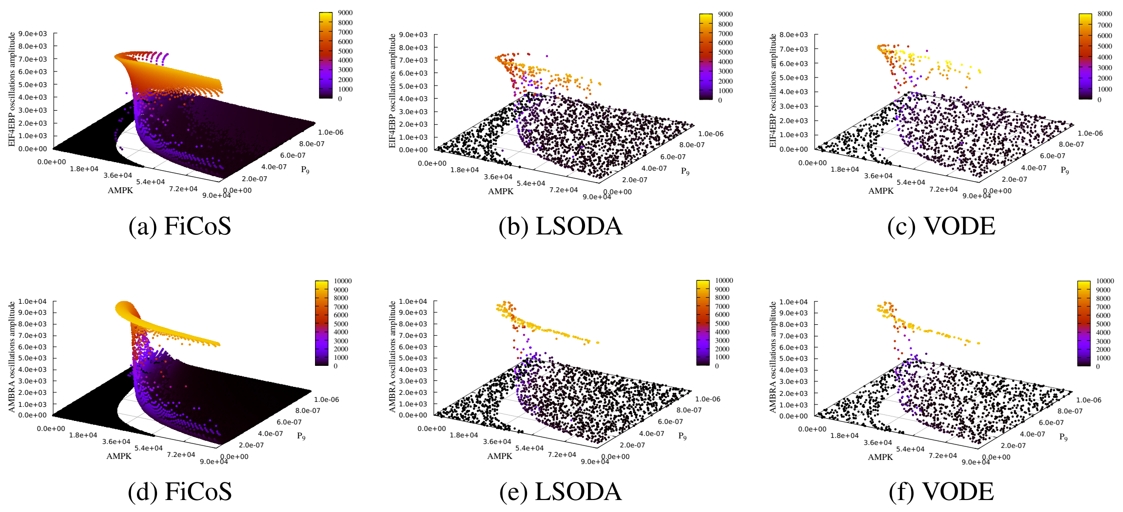


Fig. 4.34 Results of a PSA-2D on the Autophagy/Translation model by varying the initial concentration of AMPK in the range $[0, 10^4]$ (molecules/cell) and the value of the P_9 rule, which modifies 5476 kinetic parameters, in the range $[10^{-9}, 10^{-6}]$ ((molecules/cell)$^{-1}$s^{-1}). The figure shows the average amplitude of both EIF4EBP (top panels) and AMBRA (bottom panels) oscillations. Note that the dynamics are not oscillating when the amplitude value is equal to zero (black points). Considering 24 hours as time budget, FiCoS was capable of performing 36864 simulations, while exploiting LSODA and VODE we could execute only 2090 and 1363 simulations, respectively. The result of this analysis can be visually appreciated in panels (a) and (d) with respect to panels (b), (c), (e) and (f). As a matter of fact, panels (a) and (d) provide a better overview of all the paramaterizations that allow for obtaining an oscillatory regime.

and 40× faster than LSODA and VODE, respectively. Regarding the integration time, FiCoS achieved a 16× and 36× speed-up against LSODA and VODE, respectively.

In order to show the advantages provided by FiCoS in the investigation of real models, we performed a PSA-2D by varying simultaneously two parameters of this model. To be more precise, we varied the initial amount of the AMPK species and the value of the P_9 rule, which modify 5476 kinetic parameters. For both parameters, we used the sweep intervals proposed in [422], namely: (i) the range $[0, 10^4]$ (molecules/cell) for the AMPK value; (ii) the range $[10^{-9}, 10^{-6}]$ ((molecules/cell)$^{-1}$s^{-1}) for the P_9 value. The numerical values of these two parameters were determined with a uniform sampling in the Cartesian product of their ranges. We run 36864 simulations using FiCoS, partitioning the simulations in 72 batches of 512 simulations to maximize the performance of FiCoS. FiCoS completed these simulations in 24 hours, while LSODA and VODE were capable of performing only 2090 and 1363 simulations, respectively, in the same time. Figures 4.34a and 4.34d show the average amplitude of EIF4EBP and AMBRA oscillations, respectively, for each performed simulation. When the amplitude value is equal to zero (black points) the dynamics are considered non oscillating. In order to correctly detect the oscillations characterizing these molecular species, we removed the first peak that could be erroneously detected as a single oscillation.

Oscillations in the phosphorylation levels of EIF4EBP and AMBRA can generate alternating periods of autophagy and translation. Here, we verified the presence of oscillations of EIF4EBP and AMBRA at varying levels of AMPK (whose presence represents a stressed condition in the cell) and P_9, which represents the strength of the negative regulation of MTORC1 by AMPK. Figures 4.34b and 4.34e illustrate the same analysis performed using LSODA, while Figures 4.34c and 4.34f show the EIF4EBP and AMBRA oscillations obtained by VODE.

4.3.6 Treg-Teff cross regulation in multiple sclerosis

The Immune System (IS) is the ensemble of cells and molecules that protects living organisms from foreign pathogens. This complex machinery consists in a set of mechanisms whose complexity depends on the evolutionary level of the host. In mammals, besides the innate immunity, the adaptive immunity represents the most effective weapon against viruses and bacteria, thanks to its ability to specifically recognize and act against pathogens (specificity), to discriminate between self and non-self, and to remember previously encountered pathogens in order to act more rapidly (memory). While being extremely effective, adaptive immunity is not faultless. A breakdown of the mechanisms that allow the IS to discriminate between self and non-self antigens may lead to harmful effects, such as the arise of autoimmune diseases. Multiple sclerosis (MS), a disease of the Central Nervous System (CNS), falls within those.

MS is a chronic inflammatory disease that causes the removal of myelin sheath created by oligodendrocytes from axons, leading to a reduced functionality of the CNS. It is well known that a genetic predisposition correlates with MS [91]. Moreover, environmental and dietary factors may play an important role. Epstein-Barr virus (EBV) may trigger the disease onset [346, 420], while it is supposed that vitamin D could help in preventing MS [166]. Symptoms include weakness and fatigue, blurry vision, speech problems, numbness and tingling, dizziness, lack of coordination and uncontrolled bodily functions. The most common form of MS ($80-90\%$ of the total insurgence) is Relapsing-Remitting MS (RRMS) [408], where relapses (periods of disease progression) are followed by periods of remission (total or partial recovery from symptoms). RRMS usually occurs in the age of $20-40$, with a women-to-men ratio of 2:1. When left untreated, 65% of RRMS cases turn after $15-25$ years to more severe MS forms [90].

Even if the etiology of MS is not fully understood, the common shared hypothesis suggests that self-reactive T lymphocytes may be activated in the periphery by an external trigger (i.e., EBV). Activated T cells can overcome the blood brain barrier and go through the CNS [478]. Once in the brain, self-reactive cells cause inflammatory events that negatively affect both myelin and oligodendrocytes, also involving other IS entities such as B lymphocytes, macrophages, and microglia. It is worth noting that relapses usually represent the clinical correlates of inflammatory bouts. Self-reactive T lymphocytes represent one of the main actors in the development and progression of the disease, as such cells tend to decrease in the peripheral blood while increasing in the spinal fluid when relapses occur. Furthermore, homeostasis of regulatory T cells (Treg) and effectors T cells (Teff) is fundamental in preventing autoimmunity [137, 269]. More precisely, a breakdown of the peripheral tolerance mechanisms, such as the lack of functionality or deficiency of Treg functions, may bring to uncontrolled activation and proliferation of effectors T cells [382].

This hypothesis has been confirmed by Vélez de Mendizábal *et al.* [450] with the use of a model based on ODEs to reproduce RRMS. However, this model represented a very simplistic scenario, by avoiding to explicitly include the trigger of the disease represented by an external factor such as the EBV, as well as the occurrence of neural damage represented by the loss of myelin and/or the death of oligodendrocytes. Furthermore, the model totally missed to give any description of the spatial evolution of the disease. These issues were fulfilled by an agent based model (ABM) capable of better describing, from a temporal and spatial points of view, the typical shape of RRMS [337]. It must be said that, due to the significant computational efforts needed to run thousands of ABM simulations, a deeper analysis of the model parameters that may influence the disease progression was not carried out. The case study considered here refers to the cross regulation mechanism between Treg and Teff cells in RRMS. T cells are a type of white blood cells that play a central role in the human immune system. Indeed, they implement the adaptive immunity that tailors the immune response of the body to specific pathogens. T cells are commonly divided into various populations, including Cytotoxic CD8 T lymphocytes, also known as effectors T cells (Teff), the main effectors of cellular-mediated immunity that can directly attack infected or cancer cells, and CD4 T helper lymphocytes, essential to boost the immune functions by activating other immune cells. More recently, regulatory T cells (Treg) have been discovered as one of the main actors in modulating the immune system in order to maintain tolerance to self-antigens and to prevent autoimmune diseases. In particular, Treg cells are usually responsible of controlling the Teff functionalities suppressing their potentially deleterious activities. Teff cells can be inhibited by Treg cells through cell-to-cell contact and immunosuppressive cytokines. Furthermore, Treg proliferation can be stimulated as a consequence of the suppression of Teff cells. In our study we consider the activation of self-reactive Teff and Treg cells due to an EBV infection that, through a process called antigenic mimicry, misleads such cells. In this situation, in healthy people, Treg cells are able to control the spread of Teff cells activated by EBV. Instead, in diseased people a breakdown of the regulation mechanism, represented by a malfunction of Treg activities, leads to widespread inflammatory events driven by Teff cells that erroneously attack the Myelin Based Protein (MBP), a major structural component of myelin that is expressed by oligodendrocytes (ODC) in the central nervous system. This attack can irredeemably damage myelin sheath of neurons leading to the occurrence of demyelinating diseases as MS.

We performed a one dimensional PSA on the RRMS model, in which one kinetic parameter was varied within a given sweep interval (chosen with respect to a fixed reference value for each parameter). The PSA was performed by generating a set of different initial conditions—corresponding to different parameterizations of the model—and then automatically executing the parallel deterministic simulations with FiCoS. For this analysis we took into account an RBM characterized by 3200 reactions and 700 chemical species representing the RRMS model. The kinetic constant associated with the reaction involving both Treg and Teff cells was varied by taking 640 different values equally distributed in the interval $[10^{-3}, 1]$ days^{-1}. The value of this reaction rate is fundamental to describe the possible malfunction of Treg cells activities, which may lead to a breakdown of the peripheral tolerance and thus to the insurgence of the disease.

For this test we exploited a Nvidia GeForce GTX Titan Z (2880 cores, clock 876 MHz, RAM 6 GB). The results of this analysis are reported in Figure 4.35, where we can observe that for values of the kinetic constant higher than 0.109 days^{-1}, Treg cells are able to control the spread of Teff cells (top panel, yellow and purple lines) and consequently avoid the appearing of the disease. This is also visible on the ODC plot (bottom panel, yellow and purple lines) that shows how the amount of ODC, even if initially lowered due to the Teff actions, goes rapidly back to its maximum value, suggesting that any damage has been avoided or recovered at most (recoverable damage). This outcome well describes the scenario of healthy people, in which the peripheral tolerance is able to compensate for a genetic predisposition in developing the disease. For values of the kinetic constant lower than 0.109 days^{-1}, an oscillatory behavior of Teff starts to appear (top panel, red line), becoming more and more pronounced as the value of the kinetic constant decreases (i.e., top panel, black line). In this scenario, it is possible to observe that the amount of ODC decreases to around zero in correspondence to each peak in the number of Teff (top panel, red and black lines), suggesting an ongoing inflammation that causes neural damage, and thus the possibility of relapses in correspondence of each peak in the Teff amount. Interestingly, a

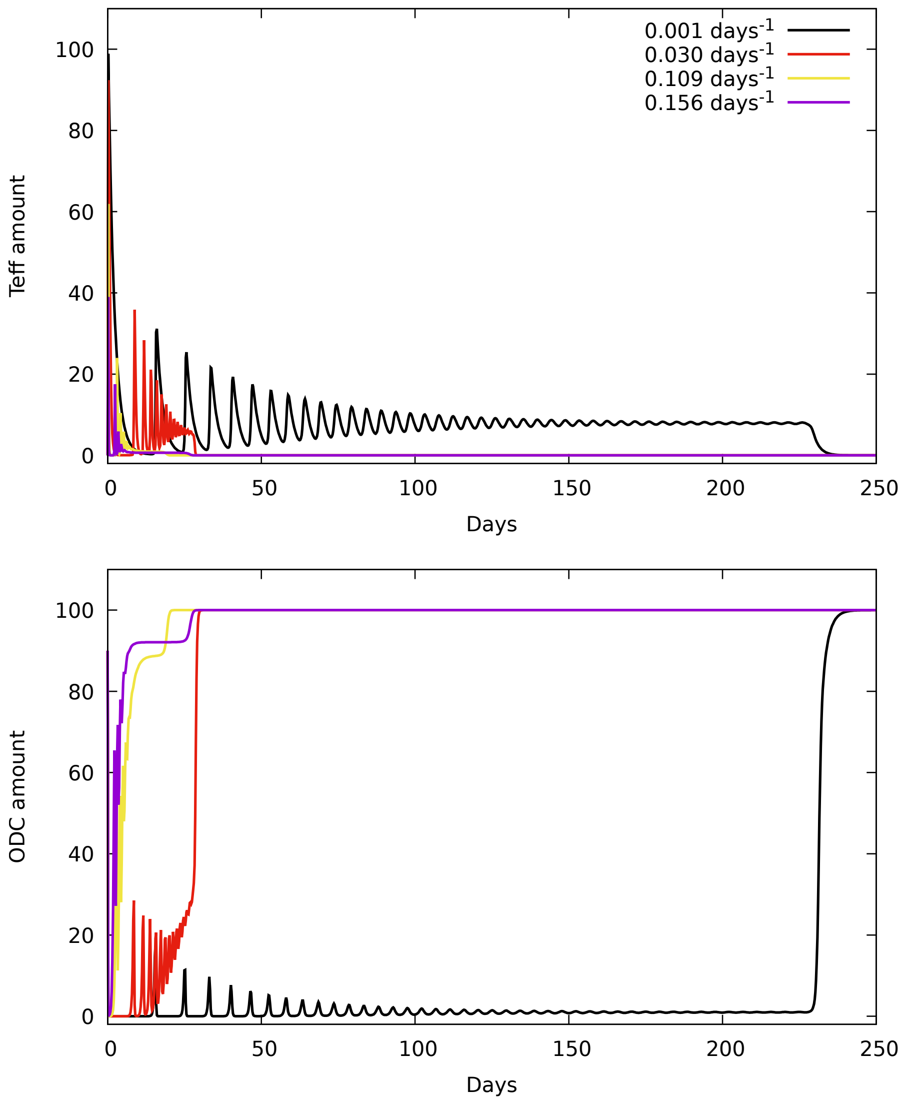


Fig. 4.35 Dynamics of Teff (top) and ODC (bottom) with different values of the kinetic constant associated with the reaction involving both Treg and Teff cells.

fixed initial quantity of EBV seems to be sufficient to start such oscillatory behavior that can be correlated to multiple relapses. This is somewhat different from the models presented in [450, 337], where each relapse was triggered by a single spread of virus.

For what concerns the computational time required to execute the PSA on the GPU, by considering the time necessary to run a single simulation with a C++ implementation of DOPRI method, exploiting a single core of a CPU Intel Core i7-6700HQ, 2.6 GHz, we estimated a speed-up around 97$\times$ on the Nvidia GeForce GTX Titan Z, thanks to the parallelization provided by FiCoS.

4.3.7 Conclusion

In order to fill the gap in the state-of-the-art of deterministic simulation, we developed FiCoS. As in the case of LASSIE, we designed FiCoS to be a “black-box” simulator able to automatically convert RBMs representing complex biochemical networks into the corresponding systems of ODEs. The performance of FiCoS was evaluated considering two sets of synthetic RBMs of increasing size, and two real models. We performed an in-depth analysis to determine which simulator should be exploited according to the number of species and reactions appearing in the model, and the number of simulations to be executed. To do this, we compared FiCoS against LASSIE, cupSODA, and two CPU-based simulators based on LSODA and VODE. Thanks to its mixed parallelization strategy, FiCoS resulted the natural choice when more than a single simulation is required and the number of species is greater than 64, noticeably outperforming all the other simulators. Afterwards, we performed a 2D PSA of a model of the Autophagy/Translation switch based on the mutual inhibition of MTORC1 and ULK1 [422], showing that FiCoS was able to execute 36864 simulations in 24 hours, while LSODA and VODE in the same time completed only 2090 and 1363 simulations. This interesting experimental finding allowed us to better understand which initial conditions lead the system to an oscillatory behavior. Considering the effectiveness of FiCoS, we realized also a PSA to investigate the effects of possible malfunctions in the Teff-Treg cross regulation mechanisms that involve a break of peripheral tolerance and bring to the occurrence of relapses in multiple sclerosis. Thanks to the acceleration provided by FiCoS, we obtained around 97$\times$ speed-up with respect to a CPU-based execution of the same analysis.

Despite the high performance that FiCoS achieved, we did not completely exploit the CUDA memory hierarchy that should be used as much as possible to obtain the best performance. Since both explicit and implicit integration algorithms are intrinsically sequential FiCoS kernels generally do not reuse any variable. Moreover, the dynamic parallelism does not allow for sharing variable among the threads belonging to the parent grid and the those inside the child grids. For these reasons, the current version of FiCoS only relies on the global memory (characterized by high latencies) and registers. We plan to develop an improved version of FiCoS tailored for models characterized by a few number of species, able to leverage both constant and shared memories: the former can be used to store the kinetic constants and the structures containing the offsets, used to correctly decode the ODEs, while the latter can be exploited to store the states of the system.

4.4 SSA accelerated on MIC coprocessors

As shown in Section 1.2.2, SSA is able to achieve an exact description of the temporal evolution of the biological system under investigation. In general, large batches of SSA runs need to be executed to determine the distribution of the system states, to collect statistically significant results, or to investigate the behavior of the system under different conditions (e.g., in the case of the Parameter Estimation problem described in Chapter 5). For computational analyses of this type, the computational burden can quickly become excessive when SSA is executed on CPUs. However, since all SSA runs can be executed independently, parallel architectures can be exploited to distribute the workload and reduce the overall running time. Algorithm parallelization is usually realized by means of multithreading [440], distributed computing on clusters [226], custom circuitry produced with Field Programmable Gate Array (FPGA) [272] or GPGPU [307, 306, 425, 74]. We highlight that these parallel technologies generally require a custom implementation of the algorithm, since most of the time the CPU code cannot be directly ported on the parallel architecture; in addition, distributed architectures need the definition of an appropriate scheduler to manage the parallel execution of processes. An alternative solution to algorithm parallelization is represented by the family of Intel Xeon Phi coprocessors, based on the MIC architecture, which allow for directly compiling and executing on the coprocessors the code implemented for CPUs (see Section 3.3).

We present here a coarse-grained implementation of SSA on MIC coprocessors, comparing its performance with respect to both CPU and GPU implementations of SSA. To this purpose we considered, on the one hand, two execution modalities of SSA for MIC: the first consisted in directly executing the same CPU source code of SSA, while the second exploited vector instructions, a peculiar capability of MIC that allows for reusing any existing CPU source code with only few modifications. On the other hand, we developed an *ad hoc* implementation of SSA for GPUs, optimizing the use of the memory hierarchy. To evaluate the running time of these sequential and parallel SSA implementations, we executed an increasing number of stochastic simulations of the Prokaryotic Gene Network (PGN) (see Appendix A.4). A family of synthetic RBMs with different size (i.e., different number of molecular species and reactions) was also used as test case to investigate the benefits of exploiting MIC vector instructions and offload capabilities. In addition to the computational time, we evaluated the costs and power consumption of the hardware employed, discussing the effort to port the existing code on parallel architectures.

Previous works already focused on the comparison of the performance of Intel Xeon Phi coprocessors against other parallel architectures, showing different results according to the specific problem under investigation. For instance, in the context of the simulation of spin systems, a comparison between Intel Xeon Phi 5110P and Nvidia Tesla K20s video card was presented in [40], highlighting that a careful implementation of the `C` code allows the MIC to compete with the GPU. On the contrary, in [130] it was shown that a Nvidia Tesla K20x outperforms an Intel Xeon Phi 5110P for the parallelization of non-bonded electrostatic computation for Virtual Screening; this work pointed out, in particular, the importance of OpenMP source code optimization. The work presented in [181] described the performance comparison among a multi-core Intel CPU, an Nvidia Tesla K20c GPU and an Intel Xeon Phi 7120P coprocessor for the execution of a tracking algorithm based on the Hough transform: the results highlighted that, in this case, the CPU performs better than both GPU and MIC coprocessors. Moreover, the authors suggested that an implementation with offloaded calculations to the coprocessors might help in achieving better performance. A multithreaded version of an algorithm to tackle the tensor transpose problem was then presented in [271]. In this case, the multi-core CPU and the MIC achieved a relevant speed-up with respect to the GPU, since the optimization of L1 cache is easier than the implementation of a coalesced global memory access on the GPU. As a final example, a comparison of the acceleration on an Intel Xeon Phi 5110P and a Nvidia Tesla K20x for protein docking calculation based on the fast Fourier transform was introduced in [396]. The GPU resulted to be 5$\times$ faster than the MIC, considering the comparable implementation costs required by these architectures.

In what follows, we show that GPU largely outperforms the other architectures, thanks to its large number of computing units, for the problem of executing increasing batches of SSA runs. Nevertheless, GPU required a complete redesign and specific programming of this simulation algorithm. On the other side, we show that MIC coprocessors allow for a relevant speed-up, especially when the simulated biochemical system is large and MIC vector instructions are used, and have the additional advantage of requiring minimal modifications to CPU code.

4.4.1 Results

We consider two different test cases of stochastic RBMs to the aim of evaluating the computational performances of MIC, CPU and GPU when executing large batches of SSA runs. The first test case is the stochastic version of PGN (see Appendix A.4), while the second test case is a family of synthetic stochastic models of increasing size, which are randomly generated according to the methodology proposed in [307]. These models are characterized by a number of species N and of reactions M ranging from (20×20) to (240×240); the values of the stochastic constants are randomly sampled with uniform distribution in $(0, 1)$. They are specifically exploited to evaluate the impact of the size of the model on the performance of the MIC architecture.

We start by showing the comparison between the computational performance achieved with Intel Xeon Phi coprocessors, CPUs and GPUs for the execution of an increasing number of SSA runs for the PGN model. Then, we investigate some specific MIC features, such as vector instructions and offload capability, to simulate a set of synthetic stochastic models with increasing size. We developed three different implementations of SSA. The first implementation was designed for the `x86` architecture, that is, the Intel Xeon Haswell E5-2630 v3 and the Intel Xeon Phi 7120P. The second implementation consisted in reusing the CPU code, modified to exploit the advantages of MIC vector instructions. The third version was specifically developed for the Nvidia Tesla K80 architecture, and optimized to fully exploit the memory hierarchy of the GPU: the state of the biochemical system is stored in the shared memory, while the matrices of stoichiometric coefficients are stored in the constant memory. All SSA implementations were written using the `C++` language and exploit the `MRG32K32a` [249] pseudorandom numbers generator [250], which is available in the Intel Math Kernel Library [456] for the CPU and MIC, and in the CURAND [317] Library for the GPU.

Simulation of the PGN model. For each architecture and its respective SSA implementation, every SSA run of the PGN model was executed for a total simulation time of $t = 80$ time units, storing the amount of all molecular species along 16 time points of the dynamics. Figure 4.36 reports the running time (in seconds) required to execute increasing batches of SSA runs on the MIC (light gray bars), MIC using vector instructions (dark gray bars), CPU (black bars) and GPU (white bars).

These results show that the CPU running time linearly increases with the number of simulations, and that the GPU largely outperforms the other architectures. In particular, the GPU running time remains almost constant while increasing the number of parallel SSA runs: this is due to the high number of cores available on the GPU used in this work, which allows for distributing the simulations over individual computing units. During this batch of tests, anyway, the GPU Nvidia K80 was far from a full usage of its computing and memory resources. Thus, although for 320 parallel SSA runs the GPU achieved a speed-up of $15\times$ with respect to the CPU, we argue that its computational performance would be even better by running a larger number of simulations. It is also worth noting that, despite the branching of CUDA threads due to the stochastic nature of all (independent) simulations, the simplicity of SSA makes coarse-grain simulation perfectly suitable for GPU architecture.

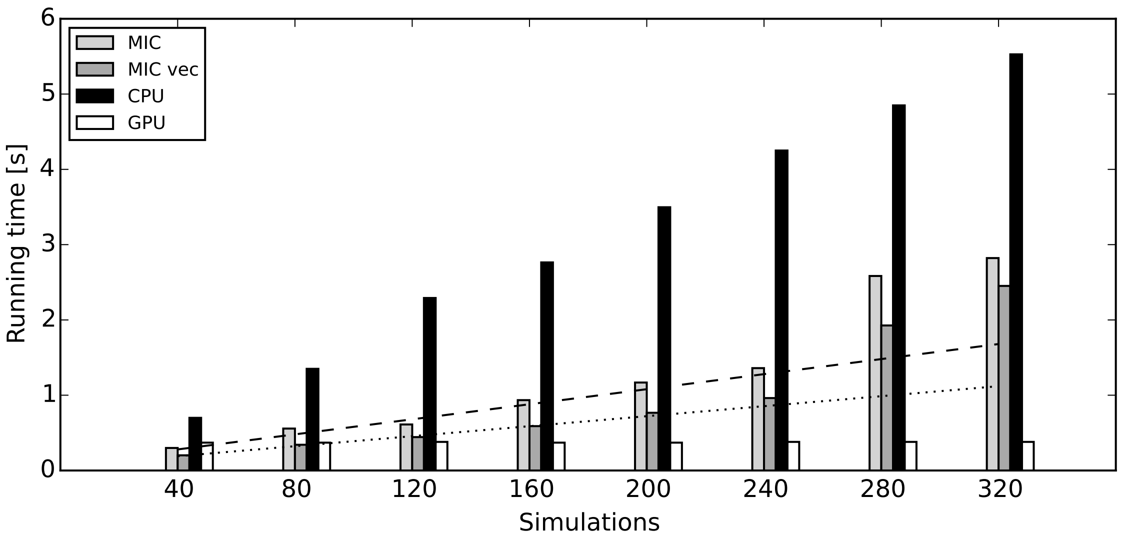


Fig. 4.36 Comparison of the running time to execute an increasing number of SSA runs of the PGN model on the three architectures: MIC (light gray bars), MIC using vector instructions (dark gray bars), CPU (black bars) and GPU (white bars). The dotted and dashed lines represent the linear regression for the estimated running time on the MIC with and without the use of vector instructions, respectively, assuming a number of cores larger than 60. The estimated values highlight the actual drop of performance when the number of parallel threads is larger than 240 (i.e., the maximum number of threads concurrently executable on the MIC). When more than 80 simulations are performed the GPU outperforms the other architectures, and the running time remains basically constant up to 320 parallel SSA runs, thanks to the large number of available cores.

In the case of MIC architecture we observed an acceleration with respect to the sequential SSA execution on the CPU. According to our results, in the case of 240 SSA runs, the speed-up achieved by this architecture—without the use of vector instructions—is $4.1\times$ with respect to the CPU (Figure 4.36, light gray bars). Since Xeon Phi coprocessors can execute up to 4 concurrent threads on each of the 60 available cores, the acceleration provided by MIC scales up to 240 simulations only. In order to highlight this limitation, we estimated by linear regression the theoretical running time of MIC to execute 280 and 320 simulations, using the running times obtained in the case of $1,\dots,240$ simulations. In Figure 4.36 we show the estimated running times with and without the use of MIC vector instructions (dotted and dashed lines, respectively). We observe that, using both SSA execution modalities on MIC, the measured running times with 280 and 320 simulations are higher than expected because of the limitation of resources.

The use of MIC vector instructions allows for an additional improvement of computational performance (Figure 4.36, dark gray bars), and highlights that this peculiar capability of

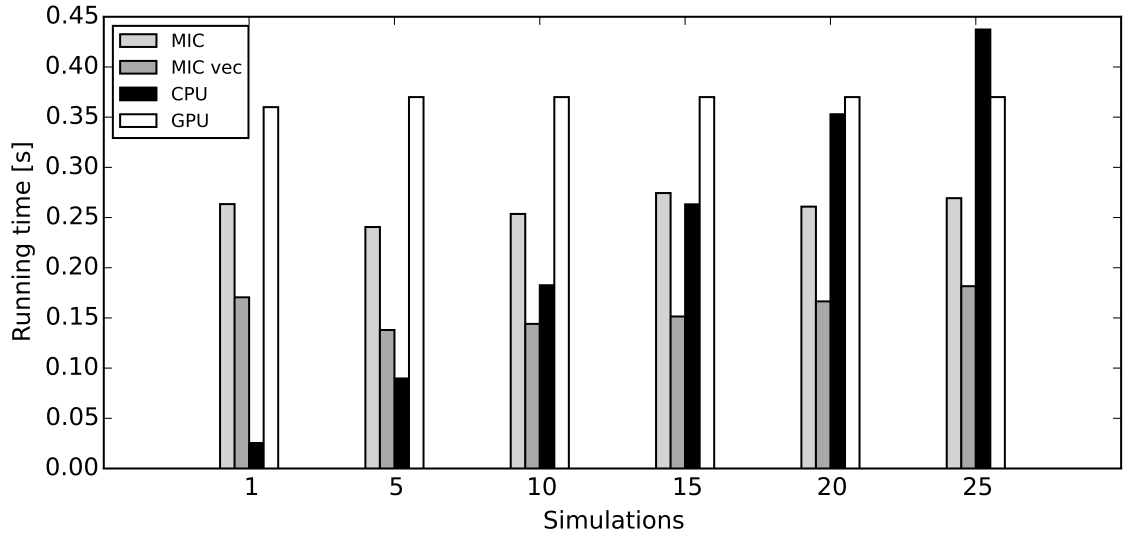


Fig. 4.37 Break-even of the running time to execute a limited number of SSA runs of the PGN model on the three architectures: MIC (light gray bars), MIC using vector instructions (dark gray bars), CPU (black bars) and GPU (white bars). When only a few simulations are executed, the CPU outperforms the other architectures thanks to its higher clock frequency. The MIC becomes advantageous with respect to CPU execution when about 10 simulations are performed, using vector instructions. Without vectorialization, the MIC is more performant than CPU when at least 15 simulations are run. Similarly, the GPU outperforms the CPU when about 20 simulations are executed.

MIC is necessary to fully leverage the computational power of Intel Xeon Phi coprocessors. More precisely, in the case of 15 simultaneous simulations (Figure 4.37), the running time of the MIC with and without the use of vector instructions changes from 0.15 to 0.27 seconds, respectively, corresponding to an overall reduction of about 42% when vector instructions are enabled. Figure 4.37 also shows the break-even between MIC, CPU and GPU. It is worth noting that when only a few SSA runs need to be executed, the CPU outperforms both GPUs and MICs, thanks to its higher clock frequency. The MIC is faster than the CPU when about 15 SSA runs are performed (10 when using vector instructions). The break-even between CPU and GPU is around 20 simulations.

Simulation of synthetic stochastic models. In the second batch of tests, we specifically focused on MIC architecture to investigate the impact of the size of the simulated model. To evaluate the performance of the Xeon Phi coprocessor, we randomly created a set of synthetic models of increasing size, i.e., 20×20, 40×40, 80×80, 160×160, whose dimensions correspond to the number of chemical species and the number of reactions, respectively.

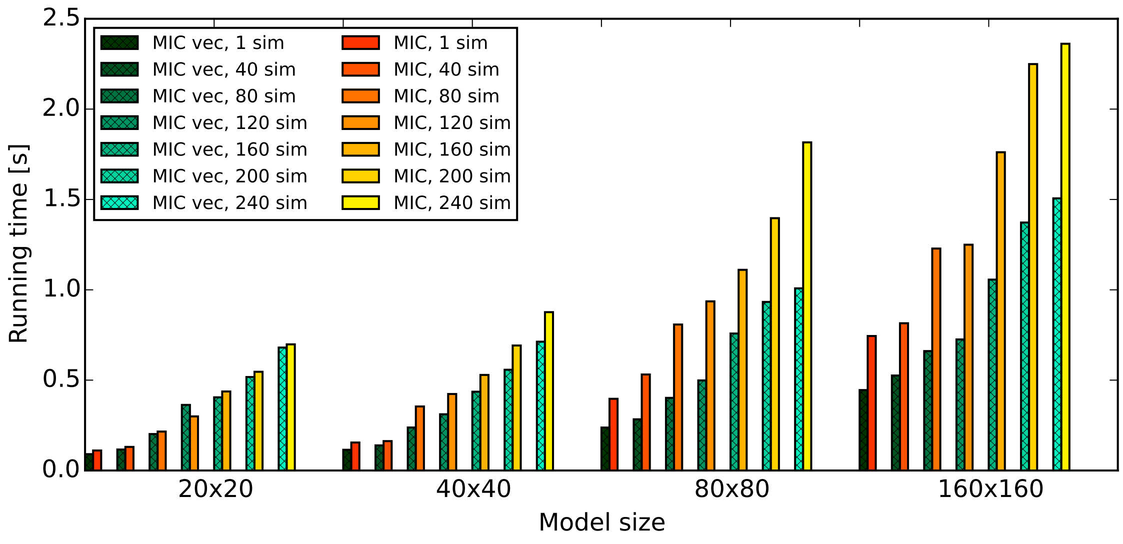


Fig. 4.38 Comparison of MIC running times with (bars with cross) and without (empty bars) the use of vector instructions to execute an increasing number of stochastic simulations (from 1 to 240) of synthetic models, having a number of species N and of reactions M equal to $(20 \times 20), (40 \times 40), (80 \times 80), (160 \times 160)$.

The models were simulated using the SSA implementation, with and without the vector instructions enabled. For every model, each SSA run was executed for a total simulation time of $t = 100$ time units, storing the amount of 10 molecular species along 11 time points of the dynamics.

The results, summarized in Figure 4.38, show two relevant trends. First, although the running time constantly increases, it is not directly proportional to the size of the model: a single simulation takes 0.11 seconds for size 20×20, and 0.74 seconds for size 160×160. This means that a biochemical system that is 8 times bigger (160×160 vs. 20×20) requires a running time of 6.7×. If we consider the case of 240 simulations, the running times are 0.6 and 1.5 seconds, respectively, corresponding to an increment of just 3.4×. The second observation is that, by using the vector instructions, the scalability is strongly improved: bars with cross in Figure 4.38 show that, in the case of 240 simulations, if we compare models of size 20×20 and 160×160, we see that the running time of the largest model is only 2.2× greater than the smallest one. In the same situation, but without using vector instructions (empty bars), the running time of the largest model is only 3.38× greater than the smallest one. Stated otherwise, the use of vector instructions achieves better computational performance when the size of synthetic stochastic models increases.

MIC offload capability. In the last test we exploited the explicit offload capabilities of the Xeon Phi coprocessors. In order to do so, we modified the SSA implementation in two ways: (*i*) we linearized all data structures, so that the arrays could be automatically transferred to the Xeon Phi; (*ii*) we added the offload compiler pre-directives, which mark the regions of the source code that must be offloaded (in our case, the initialization of the system, the calculation of the propensity functions and the update of the system states). According to our results, even in the case of "large" stochastic models of biochemical systems (e.g., 240×240 chemical species and reactions) the offload does not provide a relevant speed-up. The rationale behind this is that SSA is a relatively simple algorithm, so that the reduced number of calculations distributed on MIC cores—combined with the overhead due to MIC initialization and data transfers—affect the overall performance, making CPU more efficient for this task. Although it is possible that MIC could provide a more relevant speed-up to simulate larger models, it is usually not trivial to define stochastic models of such complexity, due to the lack of kinetic parameters and of initial molecular amounts that usually affect the availability of large scale mechanistic models of biochemical systems.

4.4.2 Conclusion

According to our results, MICs are capable of outperforming CPUs when more than 10 SSA runs are taken into account. Nevertheless, GPUs are the best choice when more than 80 SSA runs are executed. However, we highlight that some additional issues should be considered for the selection of a proper parallel architecture. Specifically, we discuss hereby also the efforts of code porting, the power consumption of the three architectures and the financial costs items.

Concerning the cost of code porting, in the case of GPGPU computing on Nvidia video cards, the effort necessary to reimplement any algorithm using the CUDA programming technique is relevant. Since this issue would increase the time-to-market, it should be seriously taken into account. On the contrary, Xeon Phi coprocessors are supposed to be fully compatible with CPUs based on the x86 instruction set; however, according to our experience, the code has to be slightly adapted in order to be correctly executed on MIC (considering the *native mode*). On the other hand, a comparison between the three architectures should also take into consideration the evaluation of costs, power consumption and theoretical peak performance. In 2016, the CPU Intel Xeon Haswell E5-2630 v3 had a cost of around €600, with a power consumption of 80W and theoretical peak performance in double precision of about 500 GFlops, which is only reachable when fused multiply-add (FMA) and advanced vector instructions (AVX) are simultaneously exploited. The characteristics of the other devices in 2016 were: €5800, 300W and 1208 GFlops for Intel Xeon Phi 7120P; €4200, 300W and 2910 GFlops for Nvidia Tesla K80. According to these data, the same theoretical peak of the Tesla K80 GPU can be achieved using either 6 CPUs or 3 Xeon Phi 7120P, with the consequent increment in terms of financial cost and power consumption. However, to fully leverage the computational power of the CPU, the implementation would require the extensive use of FMA, AVX and multithreading, requiring relevant modifications of the code.

As a final remark, we highlight that the performances of GPUs and MICs are affected by the ECC, used to avoid any error caused by natural radiations [131]. This functionality is enabled on both accelerators on the GALILEO supercomputer and introduces a relevant overhead, mainly due to bits verification. According to the tests performed by Fang *et al.*, the ECC on MIC coprocessors causes a bandwidth reduction greater than 20% [131]. Tesla GPUs exploit ECC over the whole memory hierarchy, including the global memory, L1 and L2 caches, and registers [302]. Also in the case of GPUs the bandwidth reduction is around 20% [237], so we expect that the speed-up obtained using GPUs and MIC for stochastic simulations could be further improved by disabling the ECC.

Chapter 5

Parameter Estimation of biological systems

In this chapter we present some methodologies to address the Parameter Estimation (PE) problem of biological systems, all based on global optimization approaches. In order to tackle the high computational time required by the PE problem, tools accelerated on Graphics Processing Units (GPUs) are exploited in the fitness calculation of the individuals. First, the PE of small-scale models is presented; in this context, a thorough analysis about the application of several global optimization approaches is proposed, showing that the PE problem is completely different to the classic benchmark functions [311, 432]. Afterwards, a multi-swarm PE approach [430] is proposed to deal with optimization problems characterized by multiple targets, as in the case of experimental measurements of biological systems, which are obtained under different conditions. Finally, we present the application of Fuzzy Self-Tuning Particle Swarm Optimization (FST-PSO) to the PE problem of large-scale models [442].

In what follows, we denote by P the size of the population (except for the Bat Algorithm where B indicates the number of bats), by D the number of dimensions of the search space, and by β_d^{min} and β_d^{max} (with $d = 1, \ldots, D$) the boundaries along the d-th dimension of the search space.

5.1 Fitness function definition

In this section we provide a formal definition of the fitness function used in the PE to evaluate the quality of the individuals, starting from the available target experimental data. We assume here to have a complete knowledge of the stoichiometric matrices **A** and **B** of the biological system Ω under investigation (see Section 1.1.1), as well as the initial concentrations of all molecular species.

The target data required by the PE process can be formalized as the experimental measurement of a subset of species $\mathcal{S}' = \{S_1, \dots, S_Q\} \subseteq \mathcal{S}$, with $Q \leq N$, where N indicates the total number of species in Ω. In particular for single swarm PE, a discrete-time series consisting of experimental data, sampled at a finite set of time points $\tau_1, \dots, \tau_F$, is assumed to be available for each $S_q \in \mathcal{S}'$. This set of measurements, denoted by $Y_q(\tau_f)$—which represents the concentration of the species S_q measured at time τ_f (for each $q = 1, \dots, Q$ and $f = 1, \dots, F$)—is called *Discrete-Time Target Series* (DTTS). The PE is carried out by comparing the DTTS with the corresponding simulated dynamics of the species in $\mathcal{S}'$. To be more precise, for each species $S_q \in \mathcal{S}'$ and for each time point τ_f, the fitness function is calculated as the distance between the experimental measure $Y_q(\tau_f)$ and the corresponding simulated concentrations $X_q^{\mathbf{k}}(\tau_f)$, obtained with the parameterization $\mathbf{k}$, as follows:

$$\mathcal{F}(\mathbf{k}) = \sum_{f=1}^{F} \sum_{q=1}^{Q} \frac{|Y_q(\tau_f) - X_q^{\mathbf{k}}(\tau_f)|}{Y_q(\tau_f)}. \tag{5.1}$$

$\mathcal{F}(\cdot)$ measures the relative point-to-point distance between the DTTS and the simulation dynamics, considering all the species in $\mathcal{S}'$ and all the sampled time instants τ_f, therefore assessing the quality of the candidate parameterization $\mathbf{k}$ with respect to the target DTTS. Hence, the PE problem consists in minimizing $\mathcal{F}(\mathbf{k})$ to the aim of identifying a vector $\mathbf{k}$ that provides a simulated dynamics that overlaps the DTTS.

The fitness function in Equation 5.1 has been defined considering that (*i*) this function correctly reflects the overlapping of the simulated dynamics with respect to the available experimental data; (*ii*) differently from common Root Mean Squared Error, Equation 5.1 exploits the normalization term at the denominator to accumulate relative distances instead of absolute distances, in order to prevent the most abundant chemical species to have a higher impact on the final fitness value. Notice that this fitness function represents a variant of the Mean Absolute Percent Error (MAPE) [106]. The fitness landscape shaped by MAPE is identical to that defined by Equation 5.1, although it is scaled due to the multiplicative constant used by MAPE to calculate the average of the contributions.

In the case of the typical scenario of biological laboratory research, the PE methods have to take into account as target of the PE a set of discrete-time measurements, obtained under different experimental conditions. In multi-swarm approaches, a single series of target data is assigned to each swarm and, during the optimization, the best individual of each swarm can migrate to the aim of obtaining a set of reaction constants that simultaneously fits all the experimental data. As target data for the PE task, these methods exploit a set of different experimental conditions, each one corresponding to some genetic, chemical or environmental perturbation of the system under investigation. Therefore, we assume the availability of the following experimental target data:

1. a set $\mathcal{E}$ of experimental measurements corresponding to E different initial conditions, with $E \geq 1$, which we assume to be characterized by distinct initial amounts of some molecular species in $\mathcal{S}$;
2. for each initial condition, we consider the measurement of the concentrations of a subset of species $\mathcal{S}' = \{S_1, \dots, S_Q\} \subseteq \mathcal{S}$, with $Q \leq N$, determined by means of standard laboratory technologies;
3. for each initial condition, the experimental data are assumed to be taken at some time points $\tau_1, \dots, \tau_F$, not necessarily sampled at regular intervals along the time course of the experiment.

In what follows, we denote by $Y_q^e(\tau_f)$ the amount of the species S_q, measured at time τ_f of the e-th initial condition, with $q = 1, \dots, Q$, $f = 1, \dots, F$, $e = 1, \dots, E$. As in the case of single swarm, this set of measures is called DTTS. We denote by $X_q^{\mathbf{k}_e}(\tau_f)$ the simulated concentrations of the target species S_q at time τ_f in the e-th initial condition, with $q = 1, \dots, Q$, $f = 1, \dots, F$ and $e = 1, \dots, E$, obtained by using some values $k_1, \dots, k_M$ of the kinetic constants, as specified in an arbitrary vector $\mathbf{k}_e$, where M is the number of reactions in Ω. The fitness function associated with the parameterization $\mathbf{k}_e$ is calculated as follows:

$$\mathcal{F}_e(\mathbf{k}_e) = \sum_{f=1}^{F} \sum_{q=1}^{Q} \frac{|Y_q^e(\tau_f) - X_q^{\mathbf{k}_e}(\tau_f)|}{Y_q^e(\tau_f)}. \tag{5.2}$$

The function $\mathcal{F}_e(\mathbf{k}_e)$ represents the normalized point-to-point distance between the DTTS and the simulated dynamics referring to the e-th initial condition, calculated using the amounts of all species in $\mathcal{S}'$ sampled at each time instant τ_f. Please notice that, in order to determine the vector $\mathbf{k}_e$ that provides a simulated dynamics overlapping at best the DTTS, the fitness function $\mathcal{F}_e(\mathbf{k}_e)$ has to be minimized.

5.2 Single swarm PE of small-scale models

In all the methodologies proposed below, we exploited cupSODA [306] to perform the simulations required to evaluate the quality of the individuals in the swarm that encode the putative model parameterizations. cupSODA allows for executing in a parallel fashion on the cores of the GPU all the calculations needed by this task.

5.2.1 Comparison of the performance of PPSO with PSO

In this section we compare the performances of Proactive Particles in Swarm Optimization (PPSO) and Particle Swarm Optimization (PSO) (see Section 2.3.5 and 2.3.4, respectively) for the PE of two Reaction-Based Models (RBMs): the Prokaryotic Gene Network (PGN) (see Appendix A.4) and the eukaryotic Heat Shock Response (HSR) (see Appendix A.2) [310]. The capability of PPSO (with respect to standard PSO) to correctly estimate a fitting parameterization of these two biochemical systems are assessed, in conditions characterized by an increasing number of particles and by different initialization strategy for the positions of the particles within the search space.

In all tests presented hereby, PPSO and PSO have the following settings: (*i*) damping boundary conditions; (*ii*) maximum velocity $v_d^{max} = 0.2 \cdot |\beta_d^{max} - \beta_d^{min}|$, for all $d = 1, \dots, D$; (*iii*) number of iterations $IT_{\texttt{MAX}} = 400$. In PSO we also set $c_{cog} = 2.0$, $c_{soc} = 2.0$ and inertia w linearly decreasing from 0.9 (at iteration 0) down to 0.4 (at iteration 400), which correspond to some values typically used in literature (see, e.g., [118, 388, 435, 43]).

We investigate the quality of the best model parameterization found by PPSO and PSO, and their respective computational time, by varying the number of particles in the swarm, i.e., by setting $P = 32, 64, 128, 256, 512$. The quality of the model parameterizations encoded by particles is assessed by considering the *Average Best Fitness* (ABF) over Θ independent repetitions. The ABF at iteration t is calculated as $ABF(t) = \frac{1}{\Theta}\sum_{\theta=1}^{\Theta} \mathcal{F}^{\theta,t}(\mathbf{k})$, where $\mathcal{F}^{\theta,t}(\mathbf{k})$ denotes the best fitness found during the t-th iteration of the θ-th optimization. Here, the ABF is calculated considering $\Theta = 20$ repetitions of the optimization. Since we are minimizing the fitness values, a smaller ABF corresponds to a better result.

As additional task, we test the impact on the PE outcome of two different initialization strategies, namely uniform and log-uniform (i.e., a uniform distribution in the logarithmic space) [73]. In the uniform initialization strategy the d-th component of a particle is sampled with a uniform distribution in the interval $[\beta_d^{min}, \beta_d^{max}]$. In the log-uniform initialization strategy, the d-th component of a particle is equal to $\exp\left(\ln(\beta_d^{min}) + \ln(\frac{\beta_d^{max}}{\beta_d^{min}}) \cdot \texttt{rnd}\right)$, where $\texttt{rnd}$ is a random number sampled with uniform distribution in $[0, 1)$. The log-uniform sampling is exploited here since it allows for uniformly spanning the different orders of magnitude of the search space by using a reduced set of samples, which is particularly appropriate in the case of the PE of biological systems [73]. Notice that in what follows, we refer to the log-uniform distribution as *logarithmic*.

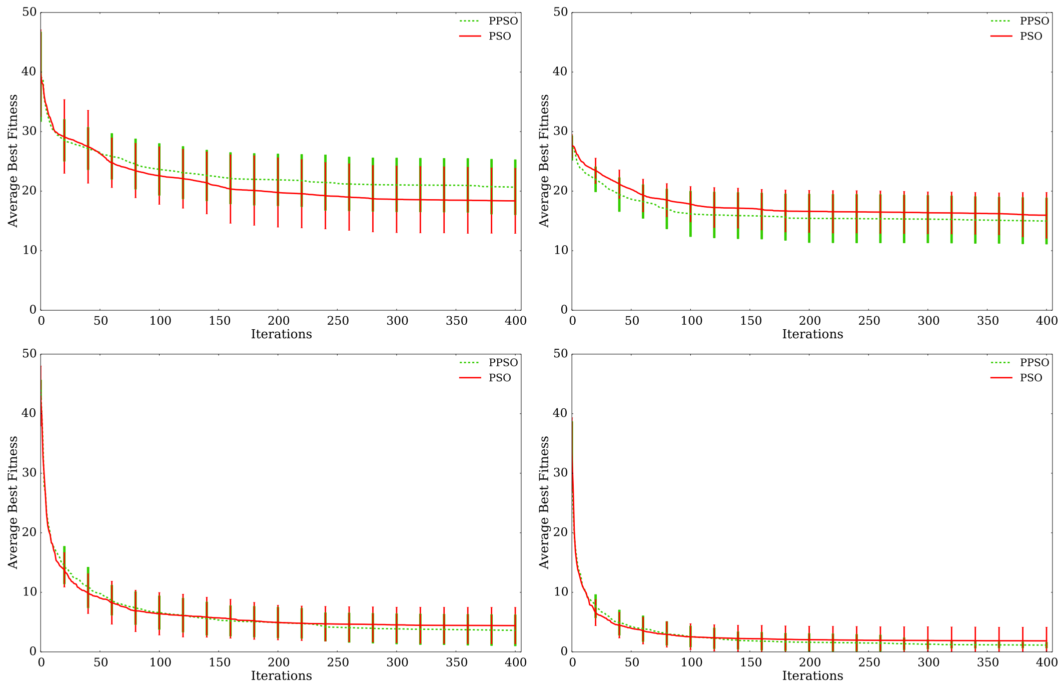


Fig. 5.1 Comparison of the ABF obtained by PPSO (green dotted line) and PSO (red solid line), using the uniform (top) and logarithmic (bottom) initialization strategies for the optimization of the PGN model. Left (right) panels correspond to a swarm with $P = 32$ ($P = 512$) particles.

PE of the PGN model

The PE of the PGN model was performed by considering $D = 8$ dimensions of the search space, with boundaries equal to $\beta_d^{min} = 1 \cdot 10^{-10}$ and $\beta_d^{max} = 100$, for each $d = 1, \ldots, D$. The DTTS for the PGN model were generated *in silico*, by sampling 10 points from a simulation realized by using a reference kinetic parameterization. The initial condition is: 500 molecules of the species DNA; zero molecules of all other species.

The first set of tests allowed us to assess the influence of the swarm size and the initialization strategy on the performance of PPSO and PSO. To this aim, we performed different PE tasks with an increasing number of particles P in the swarm, observing that with both initialization strategies the ABF decreases as the number of particles increases. For any tested value of P, our results highlighted that PPSO and PSO are characterized by a comparable convergence speed, and they reach similar results in terms of ABF at the end of the optimization process.

In Figure 5.1 we show the results obtained with $P = 32$ and $P = 512$ particles in the swarm (left and right panels, respectively). The plots clearly show that the uniform initialization

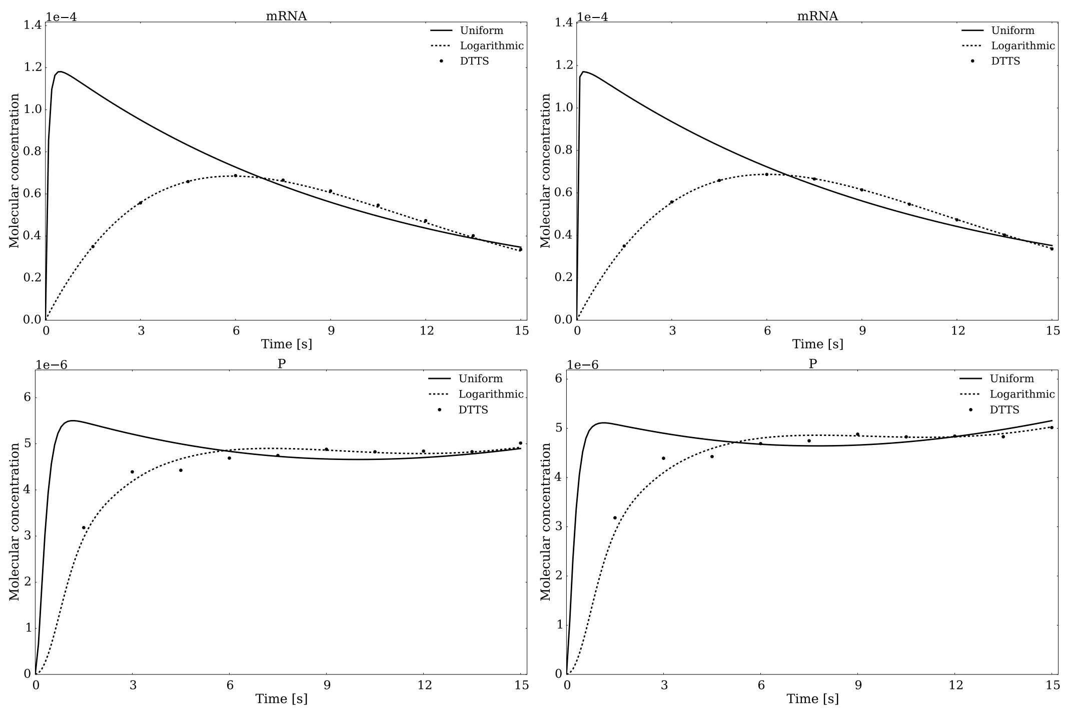


Fig. 5.2 Temporal dynamics of the species mRNA (top) and P (bottom) of the PGN model obtained with the best parameterization found by PSO (left) and PPSO (right) in the case of $P = 512$ particles. Dots represent the DTTS target used in the PE, solid (dashed) lines represent the simulation realized by using the best parameterization found during the 20 repetitions of the optimization exploiting the uniform (logarithmic) initialization strategy.

strategy (top panels) has worse performances compared to the logarithmic one (bottom panels), confirming the results obtained in [73]. As a matter of fact, with the uniform initialization strategy, both PSO and PPSO get stuck into local minima (characterized by a fitness value around 20), while the logarithmic strategy allows particles to perform a better initial exploration of the search space, therefore finding better solutions.

Figure 5.2 shows the dynamics of the species mRNA and P of the PGN model, generated by considering the best model parameterization found by PSO (left) and PPSO (right), in the case of a swarm with $P = 512$ particles. We observe that, despite the low number of points in the DTTS, our PE methodology was robust enough to converge to a good solution in the case of the logarithmic initialization. This model parameterization results in a dynamics that perfectly overlaps the target data, both in the case of PSO and PPSO.

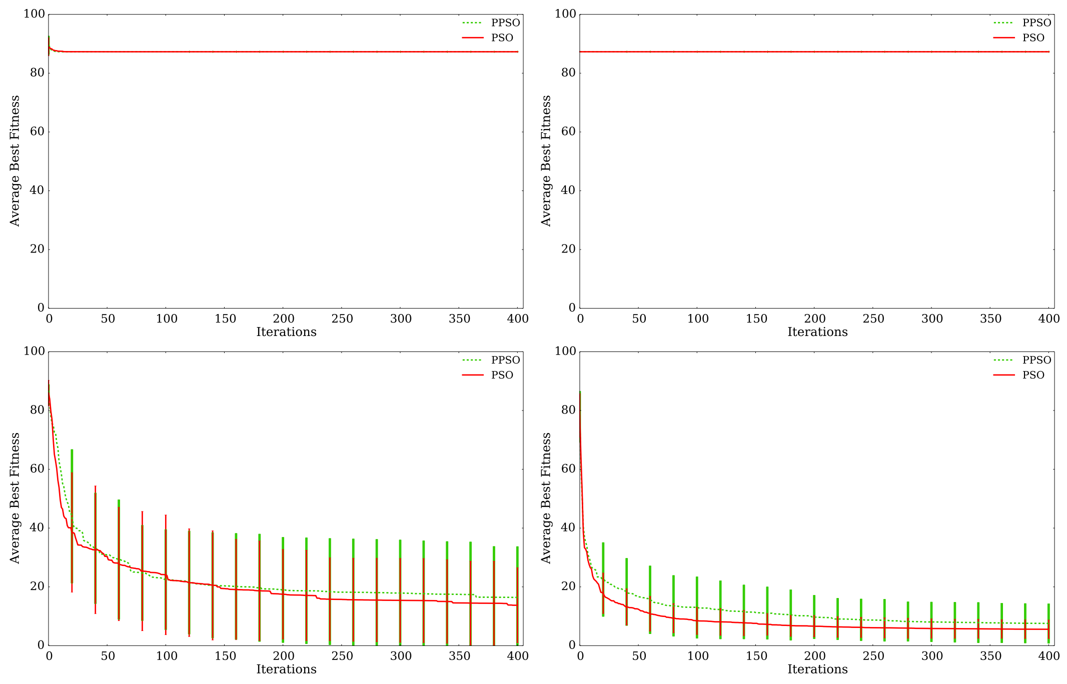


Fig. 5.3 Comparison of the ABF obtained by PPSO (green dotted line) and PSO (red solid line), using the uniform (top) and logarithmic (bottom) initialization strategies for the optimization of the HSR model. Left (right) panels correspond to a swarm with $P = 32$ ($P = 512$) particles.

PE of the HSR model

Here, we exploited the RBM of the HSR presented in [340], whereby the molecular mechanisms of HSR are formalized by means of 17 reactions among 10 species. The PE of the HSR model was performed by considering $D = 16$ dimensions, since the value of the kinetic constant of a reaction is known and it is therefore kept to its real value throughout the optimization (namely, the protein misfolding rate occurring at $42°C$). The search space boundaries were fixed to $\beta_d^{min} = 1 \cdot 10^{-9}$ and $\beta_d^{max} = 100$, for each $d = 1, \ldots, D$.

The first set of tests allowed us to assess the influence of the swarm size and the initialization strategy on the performance of PPSO and PSO for the PE of this model. Similarly to the analysis and results carried out for the PGN model, we observed that, with both initialization strategies, the ABF decreases as the number of particles increases from $P = 32$ to $P = 512$. Moreover, while the uniform initialization strategy prevented the convergence of both PPSO and PSO, the best results were obtained with the logarithmic strategy. As in the case of the PGN model, we observed that both PPSO and PSO are characterized by a

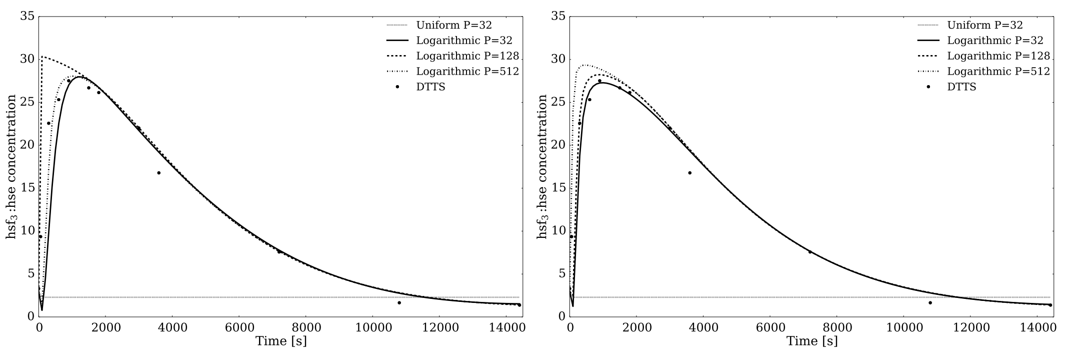


Fig. 5.4 Temporal dynamics of the hsf_3:hse species of the HSR model obtained with the best parameterization found by PSO (left) and PPSO (right). Dots represent the DTTS target used in the PE, dotted line corresponds to the best parameterization found during the 20 repetitions of the optimization exploiting the uniform initialization strategy. Solid and dashed lines refers to the best parameterization found by using the logarithmic strategy with 32, 128 and 512 particles.

similar convergence speed, and they reach similar results in terms of the ABF at the end of the optimization process.

In Figure 5.3 we show the results obtained with $P = 32$ and $P = 512$ particles in the swarm (left and right panels, respectively). With the uniform initialization strategy (top panels) both PSO and PPSO get stuck to very high values of the fitness (around 90), while the logarithmic initialization strategy (bottom panels) allows for achieving better results. In particular, in the case of logarithmic initialization and swarm size $P = 512$, although PPSO has a slightly higher ABF with respect to PSO, it is characterized by a larger variance. Thanks to this behavior, the best fitting individuals (i.e., model parameterizations with lower fitness value) were identified by PPSO. Specifically, Figure 5.4 shows the dynamics of the hsf_3:hse species of the HSR model, generated by using the best parameterization found by PSO (left) and PPSO (right). We observe that, in the case of PPSO with logarithmic initialization strategy, all simulated dynamics perfectly overlap the target DTTS. On the contrary, PSO with $P = 128$ particles and logarithmic initialization strategy obtained a parameterization that cannot fit the DTTS, especially in the first part of the dynamics. Finally, for both PPSO and PSO, the uniform initialization strategy prevented the convergence to a correct parameterization.

Analysis of the performance of PPSO

The PE shown in previous two sections, executed with an increasing number of particles, were realized to the aim of assessing the computational performance of PPSO against PSO.

For the tests, we compared the performances of a Nvidia GeForce GTX Titan Z with 2880 cores (clock 837MHz) with a Central Processing Unit (CPU) Intel Core i7-4790K (clock 4GHz).

Thanks to the GPU acceleration, the execution time with respect to the CPU was strongly reduced. In particular, we performed an experiment, tailored on the Titan Z, to execute a PE of the HSR model using swarms with $P = 2880$ particles that required 1514 seconds (approximately 25 minutes), while the execution on the CPU took 202392 seconds (approximately 56 hours), meaning that we achieved a speed-up of $133\times$. These results highlight the importance of GPU parallelization provided by cupSODA [306] for the simulations and the fitness calculations.

As a final remark, we stress the fact that cupSODA is not the only source of computational acceleration of our PE methodology. Indeed, with respect to standard PSO, also PPSO is able to reduce the overall running time. Although PPSO is computationally more expensive than PSO—due to the $\Theta(P \cdot IT_{\texttt{MAX}})$ Sugeno inferences required by the fuzzy rules—according to our tests a batch of 20 PE tasks with PPSO takes about 3 hours less than a batch of 20 PE tasks with PSO (with equal conditions). We argue that this speed-up is due to the capabilities of the fuzzy reasoner to allow the proactive particles to escape the regions characterized by a high fitness value. On the contrary, we believe that PSO is less efficient in escaping these regions, because the cognitive factor has a fixed weight in the velocity update. The regions with high fitness values are generally characterized by "extreme" model parameterizations that can lead to stiffness of the system of Ordinary Differential Equations (ODEs), therefore slowing down the simulation. This condition is automatically tackled by cupSODA, which switches to implicit integration methods. We remember that implicit integration requires multiple evaluations of the Jacobian matrix, which has a relevant computational complexity. As a consequence, PSO might require longer execution times since more particles in its swarm risk to remain stuck into bad regions of the search space.

The results obtained in this work highlight that PPSO represents a valuable alternative to classic PSO, allowing for a correct and accurate estimation of the kinetic parameters of mathematical models of biochemical systems, remarkably without the need for a fine-tuning of the functioning settings. As a matter of fact, PPSO works "out-of-the-box" by automatically selecting the inertia weight, the cognitive factor and the social factor of each particle, adjusting these settings according to the run-time performance of each particle in the swarm. In addition, the results confirm the analysis presented in [73], where the logarithmic strategy for the particle initialization was identified as more effective than the classic uniform initialization strategy for this particular application.

We will further investigate this hypothesis by analyzing and comparing the distributions of particles in the search space by using the two optimization algorithms, to possibly determine a quantitative relationship among the particle behavior, their fitness values and the overall running time.

5.2.2 Comparison of the performance of DBLA with PSO

In this section we compare the performances of Bat Algorithm with differential operator and Lévy flights trajectories (DLBA) and PSO (see Section 2.3.3 and 2.3.4, respectively) for the PE of PGN and HSR models [426].

In all tests, PSO has the following settings: (*i*) damping boundary conditions; (*ii*) maximum velocity $v_d^{max} = 0.2 \cdot |\beta_d^{max} - \beta_d^{min}|$, for all $d = 1, \dots, D$; (*iii*) number of iterations $IT_{\mathtt{MAX}} = 400$; (*iv*) cognitive factor $c_{cog} = 2.0$ and social factor $c_{soc} = 2.0$; (*v*) inertia w linearly decrementing from 0.9 down to 0.4, according to [73]. The settings used in DLBA are: (*i*) frequencies $f_{1,min} = f_{2,min} = 0$ and $f_{1,max} = f_{2,max} = 1$; (*ii*) initial loudness A^0 sampled with uniform distribution in $[1,2)$; (*iii*) initial pulse rate r^0 sampled with uniform distribution in $[0,0.1)$; (*iv*) α parameter is fixed to 0.9; (*v*) n_t parameter is fixed to 5000, as suggested in [474]. In the specific case of uniform initialization strategy, we set $f_{max} = D$, which helps to prevent premature convergence of DLBA (see Figure 5.5). In order to perform a fair comparison between DLBA and PSO, we fixed the number of iterations $IT_{\mathtt{MAX}}$ of DLBA equal to 133, since each iteration of this algorithm requires three fitness evaluations for each bat. The quality of the best model parameterization found by DLBA and PSO, and their respective computational time, are investigated by varying the number of individuals in the swarm (i.e., $P = B = 32, 64, 128, 256, 512$). We run Θ independent repetitions and considered the ABF to evaluate the quality of the model parameterizations encoded by individuals. Here, we run $\Theta = 20$ repetitions of the optimization algorithms to calculate the ABF. Furthermore, we exploited two different strategies—uniform and logarithmic—for the initial distribution of individuals in the search space to test the impact on the PE outcome.

PE of the PGN model

The PE of the PGN model considers $D = 8$ dimensions of the search space, with boundaries equal to $\beta_d^{min} = 1 \cdot 10^{-10}$ and $\beta_d^{max} = 100$ for each $d = 1, \dots, D$. The DTTS of the PGN model were generated *in silico*, by sampling 10 points from a simulation realized by using a reference kinetic parameterization.

In the set of tests performed on the PGN model, we show the influence of the swarm size and the initialization strategy on the performance of DLBA and PSO. To this aim, different PE

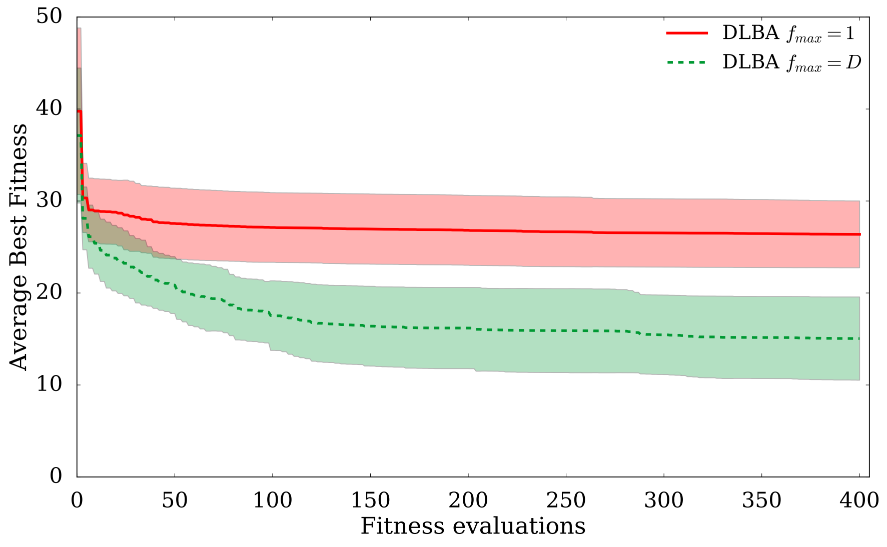


Fig. 5.5 ABF obtained by DLBA with $f_{max} = D = 8$ (green dotted line) and $f_{max} = 1$ (red solid line), using uniform initialization with $B = 32$ bats for the optimization of the PGN model. The colored areas around the ABF lines represent the standard deviation. Note that the x-axis reports the number of fitness evaluations performed by each individual in the swarm.

tasks with an increasing number of individuals $P = B$ in the swarm are performed. We observe that with both initialization strategies, when the number of individuals increases the ABF decreases. The results shown in Figure 5.6 highlight that DLBA and PSO have a comparable convergence speed and reach similar results in terms of ABF, but DLBA achieves better (worse) fitness than PSO with uniform (logarithmic) distribution. By exploiting the uniform initialization, both DLBA and PSO suffer from premature convergence; on the contrary, this issue is mitigated by the improved exploration of different orders of magnitude allowed by logarithmic sampling. In accordance with [73], these results clearly show how the logarithmic initialization employed here (right panels) allows for achieving better performance compared to uniform one (left panels).

Figure 5.7 reports the dynamics of the species P of the PGN model, obtained with the best model parameterization found by PSO (left) and DLBA (right) with $P = B = 32$ individuals. In the case of logarithmic initialization, the PE methodology converges to a good solution despite the low number of points in the DTTS, achieving a simulation of the dynamics that perfectly overlaps with the target data.

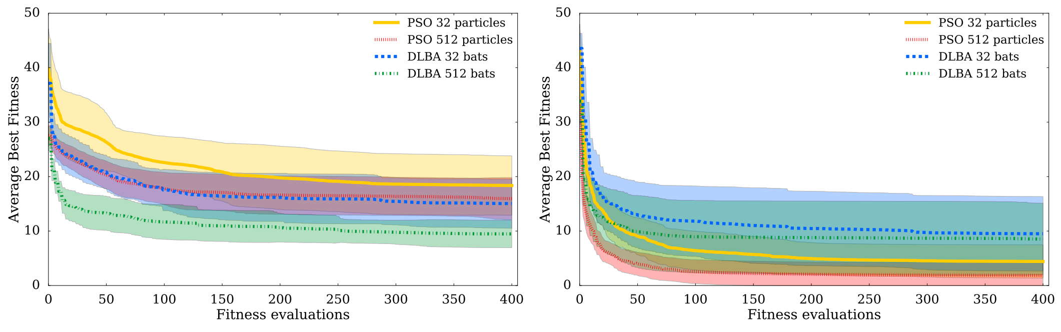


Fig. 5.6 Comparison of the ABF obtained by DLBA and PSO with $P = B = 32$ and $P = B = 512$ individuals, using the uniform (left) and logarithmic (right) initialization strategies for the optimization of the PGN model. The colored areas around the ABF lines represent the standard deviation. Note that the x-axis reports the number of fitness evaluations performed by each individual in the swarm.

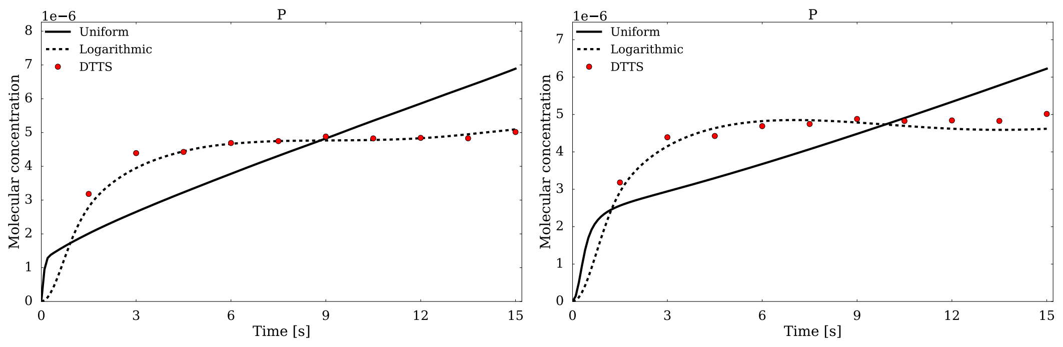


Fig. 5.7 Temporal dynamics of the species P of the PGN model obtained with the best parameterization found by PSO (left) and DLBA (right) in the case of 32 individuals. Red dots represent the DTTS used in the PE, solid (dashed) lines represent the simulation realized by using the best parameterization found during the 20 repetitions of the optimization exploiting the uniform (logarithmic) initialization strategy.

PE of the HSR model

In the set of tests performed on the HSR model, we investigated again the influence of the swarm size and the initialization strategy on the performance of DLBA and PSO. Similarly to the results obtained for the PGN model, in both initialization strategies, the ABF decreases when the number of individuals increases from 32 to 512. Moreover, while PSO suffers from premature convergence with the uniform initialization, DLBA obtains an almost good parameterization also using this initialization strategy. As in the case of the PGN model, the best results were obtained with the logarithmic strategy, whereas both DLBA and PSO are characterized by a similar convergence speed and obtain similar ABF values.

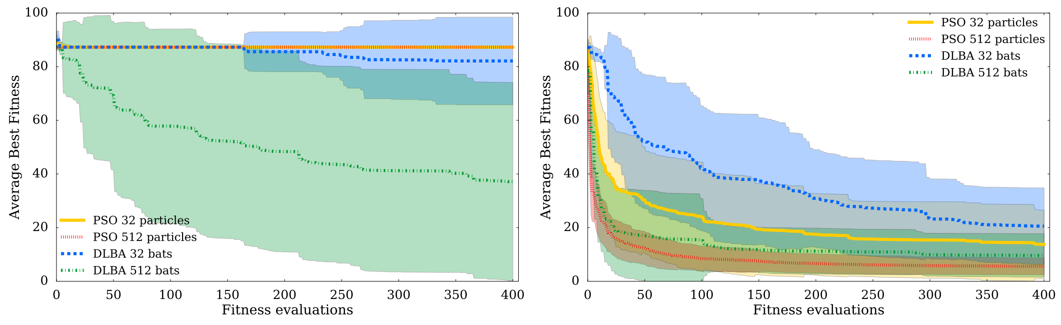


Fig. 5.8 Comparison of the ABF obtained by DLBA and PSO with $P = B = 32$ and $P = B = 512$ individuals, using the uniform and logarithmic initialization strategies for the optimization of the HSR model. The colored areas around the ABF lines represent the standard deviation. Left (right) panels correspond to uniform (logarithmic) distribution. Note that the x-axis reports the number of fitness evaluations performed by each individual in the swarm.

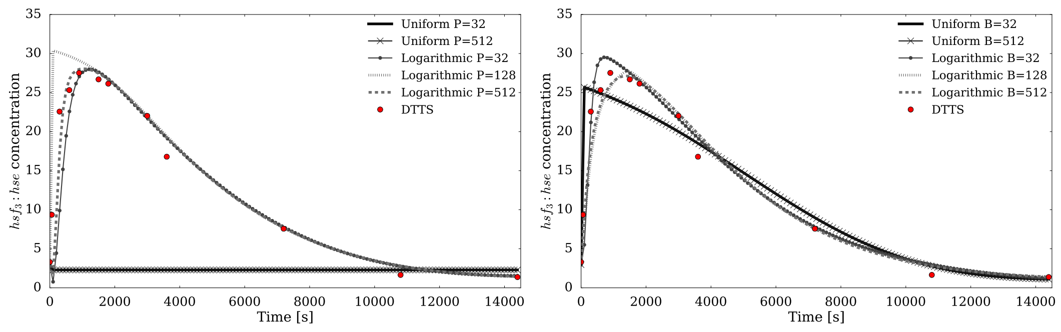


Fig. 5.9 Temporal dynamics of the species hsf$_3$:hse of the HSR model obtained with the best parameterization found by PSO (left) and DLBA (right). Red dots represent the DTTS used in the PE, solid line corresponds to the best parameterization found during the 20 repetitions of the optimization by exploiting the uniform initialization strategy. Dotted and dashed lines refers to the best parameterization found by using the logarithmic strategy with 32, 128 and 512 individuals.

In left and right panels of Figure 5.8 are shown the results obtained with the uniform and logarithmic initialization strategies, respectively. The left panel shows that PSO get stuck to very high fitness values (around 90) with the uniform initialization strategy with $P = 32$ and $P = 512$, while DLBA reaches a fairly good fitness values (around 45) with $B = 512$. Better results are achieved by both PSO and DLBA with the logarithmic initialization strategy (right panel); the best solutions found at the end of the optimization process are comparable between the two algorithms both in the case of 32 and 512 individuals. In almost all tests, DLBA is characterized by a larger variance and for this reason it is able to identify the best fitting individuals (i.e., model parameterizations with the lowest fitness value).

In Figure 5.9 are shown the dynamics of the species hsf_3:hse of the HSR model, obtained with the best model parameterizations found by PSO (left) and DLBA (right) under different conditions. Considering the logarithmic initialization strategy, both PSO and DLBA obtained parameterizations that fit the DTTS; however, in the case of $B = 32$ bats and $P = 128$ particles the first part of the dynamics is not perfectly overlapped with the DTTS. On the contrary, the uniform initialization strategy prevented the convergence to an optimal parameterization in the case of PSO, while DLBA is able to identify a set of kinetic values that fits the DTTS.

Analysis of the performance of DLBA

The aforementioned PEs were executed with an increasing number of individuals to assess the computational performance of DLBA against PSO. The GPU acceleration allowed us to strongly reduce the execution time with respect to the CPU. For the tests, a Nvidia GeForce GTX Titan Z (2880 cores, clock 837MHz) was compared with a CPU Intel Core i7-4790K (clock 4GHz). To show the importance of parallelization of the simulations and the corresponding fitness calculations, we performed an experiment that saturates the Titan Z. In this task, we executed a PE of the HSR model using swarms with $B = 2880$ bats, achieving a speed-up of $61\times$ since the PE on the CPU required around 83 hours, while the execution on the GPU required around 80 minutes. Besides cupSODA, DLBA itself is able to reduce the overall running time, with respect to PSO. Indeed, the efficiency of a metaheuristic search algorithm is based on balancing between intensification (exploitation) and diversification (exploration). Lévy flights generate new solutions that allow for escaping from local minima, i.e., regions of the search space with high fitness values. These regions are characterized by solutions that can lead to ODE stiffness, slowing down the simulations since cupSODA switches to implicit integration methods. Avoiding these regions, DLBA takes about 10 hours less than PSO (with equal conditions) to execute a batch of 20 different optimizations for the PE problem.

According to our results, DLBA represents a valuable alternative to classic PSO. In our tests, DLBA outperformed PSO when the uniform initialization is employed. However, with both algorithms, the logarithmic initialization improves the ABF, confirming the results presented in [73].

5.2.3 Reboot strategies in PSO

In this section, three different reboot strategies for PSO are presented [409, 410], whose aim is to reinitialize particle positions to avoid particles to get trapped in local optima. We compare the performance of PSO coupled with the reboot strategies with respect to standard PSO in the case of the PE of two biochemical systems, namely, the HSR model and the Ras/cAMP/PKA signaling pathway in yeast (see Appendix A.5).

Over the years, different works presented modifications aimed to improve the standard PSO algorithm. Some of them introduced specific strategies that attempted to overcome stagnation of the swarm in local minima, either by implementing hybrid PSO approaches [332, 453, 323] or reboot strategies that re-initialize all particles in the swarm [146, 107] in the case of problems characterized by very large search spaces. Here, we define different reboot strategies in which the particles are re-initialized if they are too close to the global best position found so far by the swarm, or if the global or local best positions remain unchanged for a given number of iterations.

Reboot strategies in PSO are introduced to avoid early convergence of the swarm and stagnation of particles in local minima. An efficient reboot strategy should accurately identify the particles that are no longer exploring effectively the search space, and re-initialize their positions to improve diversity in the swarm and possibly achieve a better exploration. Different reboot strategies were previously integrated in PSO [146, 107] to deal with the problem of continuous optimization in large search spaces. In this context, a "restart" of the algorithm is implemented by re-initializing all particles in the swarm except for the global best position found so far.

We designed three different reboot strategies and assessed their impact on the performance of PSO applied to the PE of the kinetic constants of biochemical systems. In particular, our three reboot strategies, namely *Global*, *Local* and *Distance*, are defined as follows:

- *Global*: if the global best position $\mathbf{g}$ remains unchanged for η iterations, where η is a user-defined parameter, the position $\mathbf{x}_i$ is re-initialized by sampling from a logarithmic distribution in $(\beta_d^{min}, \beta_d^{max})$, the local best position $\mathbf{b}_i$ is set to $\mathbf{x}_i$ and the velocity $\mathbf{v}_i$ is reset to $\mathbf{0}$, for each particle i of the swarm;

- *Local*: if the local best position $\mathbf{b}_i$ of a particle i does not change for η iterations, its position $\mathbf{x}_i$ is re-initialized by sampling from a logarithmic distribution in $(\beta_d^{min}, \beta_d^{max})$, its local best position $\mathbf{b}_i$ is set to the new position and its velocity $\mathbf{v}_i$ is reset to $\mathbf{0}$;

- *Distance*: at each iteration, if $\|\mathbf{b}_i^* - \mathbf{g}^*\| < \theta$, the position $\mathbf{x}_i$ of the particle i is re-initialized by sampling from a logarithmic distribution in $(\beta_d^{min}, \beta_d^{max})$, its local best

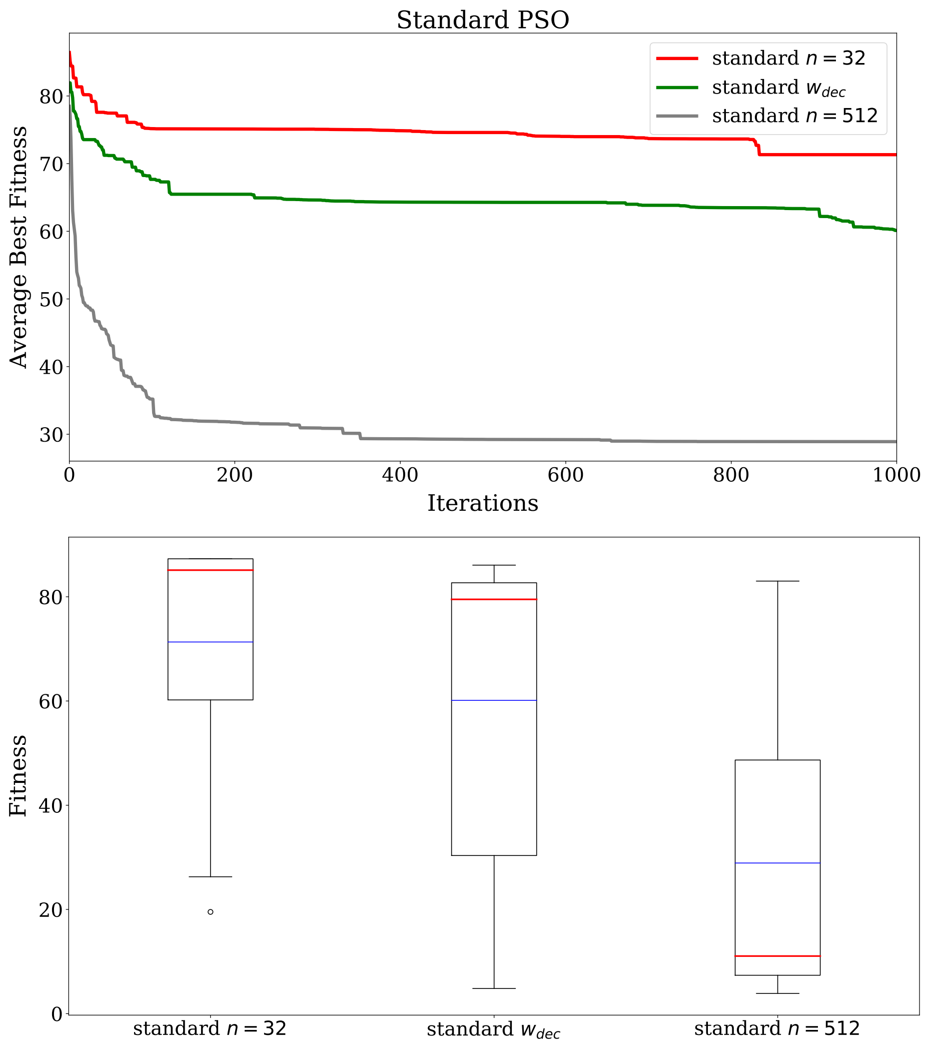


Fig. 5.10 *Top*: ABF comparison of the PE executed on the HSR model with PSO having swarm size $n = 32$ and inertia $w = 0.729$ (red line), swarm size $n = 512$ and decreasing inertia w (green line), and swarm size $n = 512$ and inertia $w = 0.729$ (gray line). *Bottom*: boxplots obtained considering the best fitness values reached by PSO during the last iteration of each optimization, with the same settings specified above. The red line corresponds to the median, while the blue line corresponds to the mean of the distribution.

position $\mathbf{b}_i$ is set to the new $\mathbf{x}_i$ and its velocity $\mathbf{v}_i$ is reset to $\mathbf{0}$. θ is a user defined parameter, $\mathbf{b}_i^* = (\log_{10}(b_{i,1}), \ldots, \log_{10}(b_{i,D}))$ and $\mathbf{g}^* = (\log_{10}(g_1), \ldots, \log_{10}(g_D))$ are the logarithmic transformation of $\mathbf{b}_i$ and $\mathbf{g}$, respectively.

In all tests presented hereby, PSO has the following settings: (*i*) damping boundary conditions; (*ii*) maximum velocity $v_d^{max} = 0.2 \cdot |\beta_d^{max} - \beta_d^{min}|$, for all $d = 1, \ldots, D$; (*iii*) number of iterations $IT_{\texttt{MAX}} = 1000$. We also set $c_{cog} = c_{soc} = 1.494$, which correspond to values used in literature (see, e.g., [443]). Moreover, we compared the effect of decreasing and fixed values for the inertia parameter w. In particular, we used an inertia decreasing from $w = 0.9$ to $w = 0.4$, and a fixed inertia value equal to $w = 0.729$ [443].

The performances of the different optimization strategies are assessed by considering the ABF over Θ independent repetitions of the PE. Here, the ABF is calculated over $\Theta = 20$ repetitions of the optimization process.

PE of the HSR model

In the PE of the HSR model performed here, the DTTS were generated *in silico* by sampling 140 points from a simulation realized using a reference kinetic parameterization. In the first set of tests, we assessed the influence of the swarm size and of the inertia updating strategy on the standard PSO algorithm. In Figure 5.10 we show the results obtained with a swarm size $n = 32$ and a fixed inertia $w = 0.729$, swarm size $n = 512$ and a linearly decreasing inertia (from $w = 0.9$ to $w = 0.4$ throughout the entire simulation), and swarm size $n = 512$ and inertia $w = 0.729$. These results show that a larger number of particles produce lower ABF values (see Figure 5.10, top) and reach better results at the end of the optimization process with respect to a swarm of smaller size, as highlighted in the boxplots (Figure 5.10, bottom). Moreover, Figure 5.10 shows that PSO with a fixed inertia outperforms PSO with a decreasing inertia in the PE of this biochemical model, despite it has been previously shown that a decreasing inertia allows for achieving better results on benchmark functions.

Figures 5.11, 5.12 and 5.13 show the results obtained by PSO implementing our three different reboot strategies. In these tests we varied the threshold parameters exploited by the reboot strategies, i.e., η for *Global* and *Local*, θ for *Distance*. While the performance of the *Distance* reboot (Figure 5.13) appears to be strongly affected by the choice of θ values, both *Local* (Figure 5.12) and *Global* (Figure 5.11) reboot strategies are robust with respect to the choice of η values, as also indicated by the smaller dispersion in the boxplots. The threshold parameters directly influence the number of reboots that take place at each iteration of PSO, even though this does not necessary lead to better performance. For instance, by looking at Figure 5.13 it is evident how $\theta = 0.1$ performs better than $\theta = 0.01$. Moreover, the latter is even worse than in the case of $\theta = 0.25$. Thus, it is necessary to find a good trade-off for the threshold parameter, in order to prevent both excessive rebooting (i.e., particles are distant and should not be rebooted, yet) and an insufficient rebooting (i.e., particles never actually get that close to $\mathbf{g}$, so that the reboot is never triggered).

Afterwards, we selected the best threshold values of each reboot strategy and compared them against standard PSO. The results, presented in Figure 5.14, show that each of the three strategies outperforms PSO both in terms of ABF and interquartile range of the final best

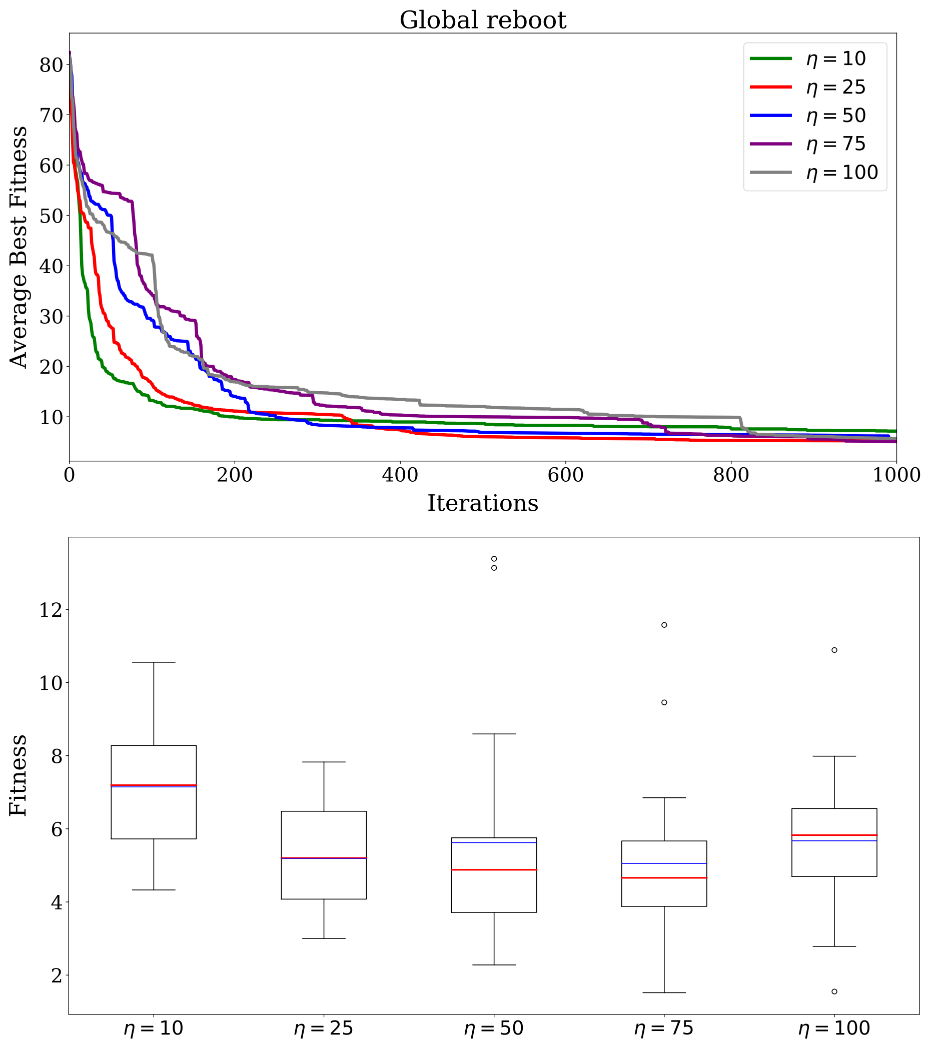


Fig. 5.11 *Top*: ABF comparison of the PE executed on the HSR model with PSO implementing the *Global* reboot strategies. Different colors correspond to different threshold values. In all cases, the inertia value is set to $w = 0.729$ and the swarm size to $n = 512$. *Bottom*: boxplots obtained considering the best fitness values reached by PSO coupled with the *Global* reboot, during the last iteration of each optimization, with the same settings specified above. The red line corresponds to the median, while the blue line corresponds to the mean of the distribution.

fitness. In particular, the *Global* and *Local* reboot procedures obtain a better final ABF and a very low statistical dispersion, as shown in the boxplots of Figure 5.14.

Since the *Local* strategy seemed to consistently outperform the other approaches, we tested an additional reboot strategy: instead of sampling the new positions from a logarithmic distribution, we used a lognormal distribution with mean equal to $(\log(\beta_d^{min}) + \log(\beta_d^{max}))/2$ and σ equal to $(\log(\beta_d^{max}) - \log(\beta_d^{min}))/2$ [264]. In Figure 5.14 we denote the *Local* strategy

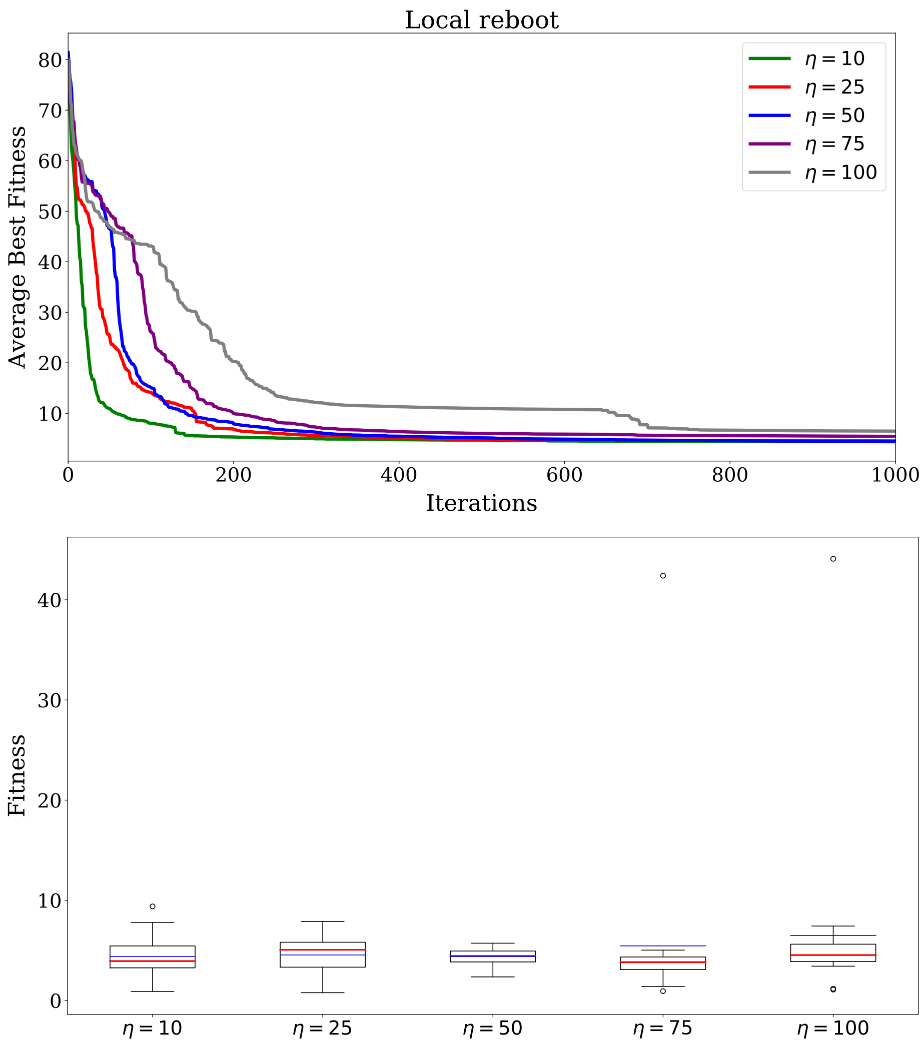


Fig. 5.12 *Top*: ABF comparison of the PE executed on the HSR model with PSO implementing the *Local* reboot strategies. Different colors correspond to different threshold values. In all cases, the inertia value is set to $w = 0.729$ and the swarm size to $n = 512$. *Bottom*: boxplots obtained considering the best fitness values reached by PSO coupled with the *Local* reboot, during the last iteration of each optimization, with the same settings specified above. The red line corresponds to the median, while the blue line corresponds to the mean of the distribution.

based on logarithmic samples and lognormal samples with $\texttt{local}_{loguni}$ and $\texttt{local}_{lognorm}$, respectively. According to our results, the new version slightly improves the average performance (top plot) although with a higher variance in the final ABF (bottom plot). Moreover, the generation of lognormally distributed random deviates is more computationally demanding than uniform random numbers (approximately five times slower). Due to these drawbacks, $\texttt{local}_{loguni}$ remains the most reliable strategy for PSO reboots.

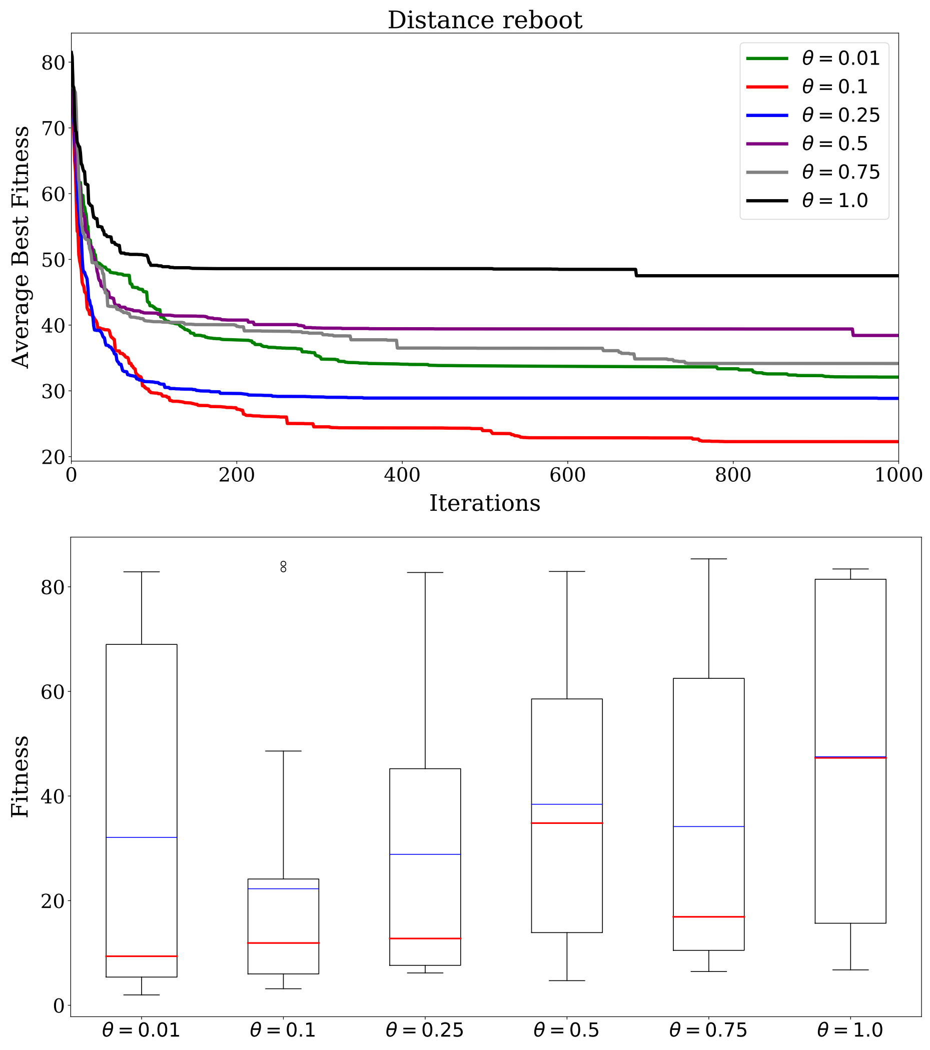
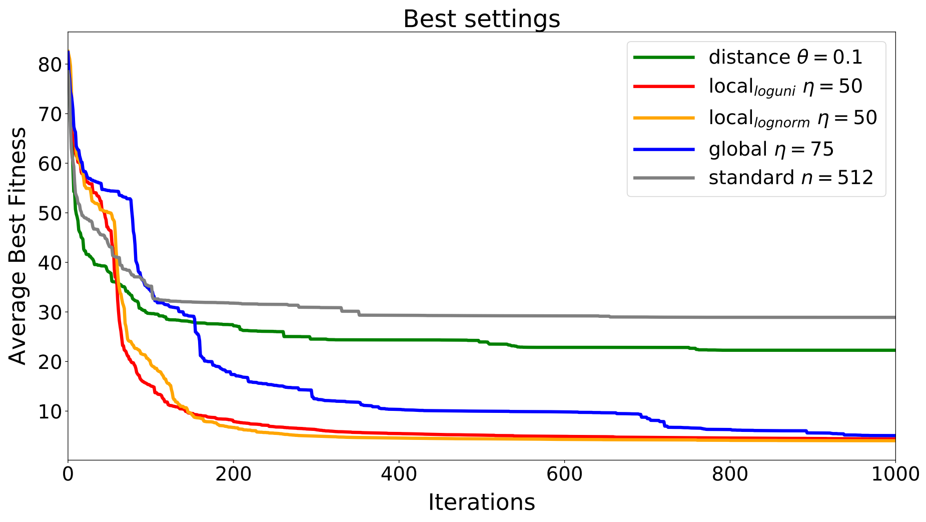


Fig. 5.13 *Top*: ABF comparison of the PE executed on the HSR model with PSO implementing the *Distance* reboot strategies. Different colors correspond to different threshold values. In all cases, the inertia value is set to $w = 0.729$ and the swarm size to $n = 512$. *Bottom*: boxplots obtained considering the best fitness values reached by PSO coupled with the *Distance* reboot, during the last iteration of each optimization, with the same settings specified above. The red line corresponds to the median, while the blue line corresponds to the mean of the distribution.

Finally, to verify the "goodness" of the PE results, we simulated the dynamics of the HSR model by using the best parameterization found by PSO (over the 20 optimization runs) with *Local* and *Global* reboot strategies. The temporal dynamics of the hsf3:hse species depicted in Figure 5.15 shows that both the *Local* and *Global* reboot found a parameterization that allows for closely approximating the target dynamics.



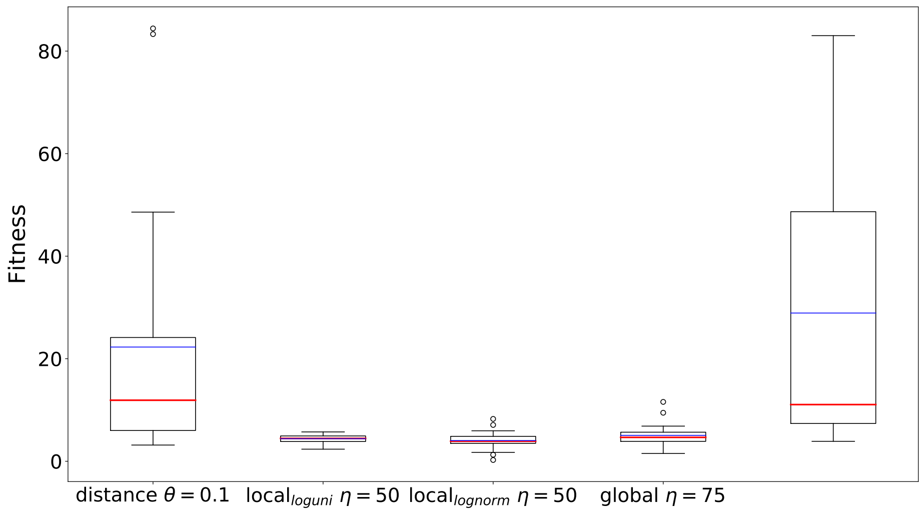


Fig. 5.14 *Top*: ABF comparison of the PE executed on the HSR model with standard PSO ($n = 512$ and $w = 0.729$), and PSO implementing the three different reboot strategies, performed with their respective best parameter threshold. *Bottom*: boxplots obtained considering the best fitness values reached by PSO and PSO coupled with the reboot strategies, during the last iteration of each optimization, with the same settings specified above. The red line corresponds to the median, while the blue line corresponds to the mean of the distribution.

PE of the Ras/cAMP/PKA signaling pathway

Here, we rely on the model of the Ras/cAMP/PKA signaling pathway (see Appendix A.5), formalized by means of 35 reactions among 30 molecular species. The PE was performed by considering $D = 35$ dimensions, and the search space boundaries were fixed to $\beta_d^{min} = 10^{-9}$ and $\beta_d^{max} = 10$, for each $d = 1, \ldots, D$. The DTTS was generated *in silico*, by sampling 1000 points from a simulation realized by using a reference kinetic parameterization.

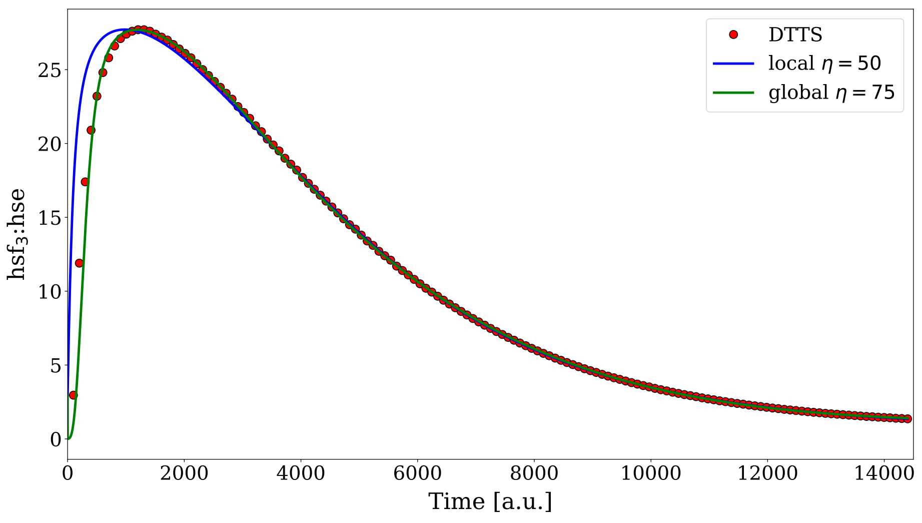


Fig. 5.15 Temporal dynamics of the hsf$_3$:hse species of the HSR model, obtained with the best parameterization found by PSO coupled with the *Local* ($\eta = 50$) and *Global* ($\eta = 75$) reboot strategies.

The size of this model, compared to the HSR model, further increases the complexity of the PE task. Thus, to the aim of estimating the kinetic parameters of the Ras/cAMP/PKA model, we exploited the reboot procedure that performed best in the previous tests, namely, the *Local* reboot strategy with $\eta = 50$. Figure 5.16 shows the target data and the simulated dynamics of the cAMP species obtained by using the best parameterization found by standard PSO and PSO coupled with the *Local* reboot strategy. We observe that, on the one hand standard PSO was unable to find a correct parameterization; on the other hand, PSO coupled with the *Local* reboot strategy identified an appropriate parameterization of the Ras/cAMP/PKA model, achieving a dynamics that fits with the target data.

Conclusive remarks

The results presented in the previous sections highlight that a fixed inertia parameter seems to be more convenient in the case of the PE problem. Moreover, we showed that reboot strategies can be coupled with PSO to avoid early convergence and stagnation of the swarm, thus achieving a correct and accurate estimation of the kinetic parameters of medium-size mathematical models of biochemical systems (characterized by tens of chemical reactions).

In particular, the *Local* reboot strategy proved to be remarkably reliable, since it is characterized by *(i)* a better convergence speed; *(ii)* lower ABF with respect to the other considered strategies; *(iii)* a higher robustness with respect to different choices for its functioning setting η.

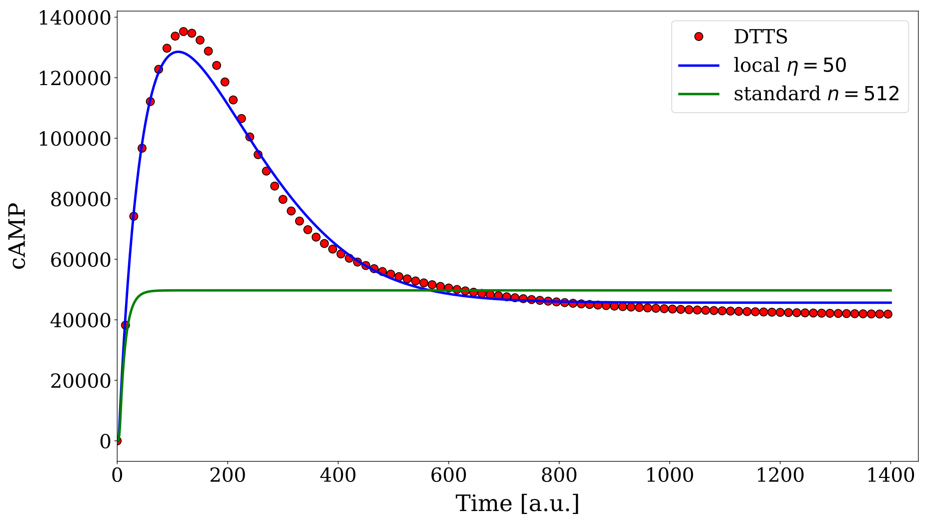


Fig. 5.16 Temporal dynamics of the cAMP species of the Ras/cAMP/PKA model, obtained with the best parameterization found by standard PSO and PSO coupled with the *Local* ($\eta = 50$) reboot strategy.

Moreover, we will apply the PE methodology based on PSO and cupSODA to more complex models of biological systems, where standard approaches fail because of the computational burden of the simulation process. Finally, when the biochemical systems under investigation are characterized by molecular species present in small quantities, stochasticity plays a fundamental role and cannot be neglected. In these situations, stochastic simulation algorithms must be employed to calculate the fitness of PSO particles; we will therefore replace cupSODA with cuTauLeaping [307]. We will also develop an appropriate fitness function able to deal with complex and noisy dynamics, based on some statistical measure computed according to the outcome of different repetitions of the stochastic simulations. ❙

5.2.4 Comparison between PE and benchmark functions

In this section we show that the state-of-the-art optimization methods, able to largely outperform other metaheuristics on benchmark functions, can be characterized by considerable poor performances when applied to real-world problems, without considering their specific peculiarities, like the PE.

Generally, the performances of these metaheuristics are assessed by relying on different sets of benchmark functions, specifically designed to test the search capabilities of the various optimization strategies [215]. However, when the same algorithms are applied to real-world problems pertaining to different application domains, which can involve continuous or discrete optimization tasks, their performances may considerably change. For instance, this phenomenon was observed in the case of the optimization of atomic and molecular clusters [112], building energy systems [221] and aircraft design [93]. This outcome is coherent with the *no free lunch theorem* [471, 273], which states that no algorithm outperforms all the competitors in any optimization problem, which imply that techniques with outstanding performances in a benchmark test suite could not properly work on every real-world problems [267]. For instance, Da Ros *et al.* [96] compared stochastic optimization methods (i.e., Artificial Bee Colony (ABC), Differential Evolution (DE), PSO, and Simulated Annealing) for the estimation of kinetic parameters of a biochemical model for alcoholic fermentation in bioreactors. Their results show that benchmark functions are not representative of real optimization problems, and the evaluation of global optimization metaheuristics based on these benchmark functions alone may induce strong biases.

Here, we exploit six well-known benchmark functions to compare the performances of ABC, Covariance Matrix Adaptation Evolution Strategy (CMA-ES), DE, Estimation of Distribution Algorithm (EDA), Genetic Algorithms (GAs), PSO, and its fuzzy-based settings-free variant FST-PSO (see Chapter 2 for further details). Then, we apply the same algorithms to the PE problem by using a set of synthetic biochemical models of increasing size (25 or 50 molecular species and reactions). By measuring the convergence speed and the quality of the final results achieved by the different metaheuristics, we show that algorithms able to perform efficiently on benchmark functions can be completely unfit for the PE problem. In particular, to investigate the performance of the optimization algorithms listed above, we exploited the computational tool described in Section 4.1 to randomly generate 12 different instances of RBMs of increasing size (6 RBMs are characterized by 25 reactions and species, 6 RBMs are characterized by 50 reactions and species). Each RBM satisfies the following characteristics:

- the initial concentrations of the molecular species are sampled from a logarithmic distribution in the interval $[10^{-6}, 1)$;
- the values of the kinetic constants are sampled from a logarithmic distribution in the interval $[10^{-8}, 10]$;
- only zero, first and second-order reactions (i.e., $\text{max}_{\text{ord}} = 2$, meaning that at most 2 reactant molecules of the same or different species can appear in a reaction), are used to create the stoichiometric matrix $\mathbf{A}$;
- the stoichiometric matrix $\mathbf{B}$ is created using at most 2 product molecules for each reaction (i.e., $\text{max}_{\text{prod}} = 2$).

We highlight that we exploited a logarithmic distribution since the concentrations and kinetic constants of biochemical systems generally span over multiple orders of magnitude [485, 73]. Deterministic simulations of RBMs and the fitness function evaluations were executed in parallel by offloading the calculations onto the GPU by means of cupSODA.

Benchmark function suites for numerical optimization

Several benchmark function suites have been proposed in the literature to evaluate the performance of global optimization techniques, since real-world problems cannot be straightforwardly exploited to this purpose. Indeed, real-world problems are often characterized by additional complex features that basic optimization algorithms might not be able to handle [141]. Considering the specific case of real-parameter numerical optimization, every year research competitions are organized by the IEEE Congress on Evolutionary Computation (CEC) [92] and the Genetic and Evolutionary Computation Conference (GECCO). Among them, it is worth mentioning the workshop on Real-Parameter Black-Box Optimization Benchmarking (BBOB) [148] that exploits the COmparing Continuous Optimisers (COCO) benchmarking platform [183]. In addition, there exist real-world problems that have intrinsic discrete structure and solutions; in this context Doerr *et al.* [113] improved the COCO software by introducing pseudo-Boolean optimization problems, providing an environment to empirically analyze and evaluate the performance of pseudo-Boolean black-box heuristics. A generalized and dynamic benchmark generator was also proposed in [256] to construct dynamic environments in the binary, real and combinatorial spaces.

Even though many different benchmark functions were proposed for continuous optimization (e.g., global optimization, dynamic optimization, multimodal optimization), a complete and unified framework for generating benchmark functions able to take into account different properties pertaining to real-world problems has not been yet proposed. Recently, an attempt in this direction was made by Li *et al.*, who presented a novel framework that aims at constructing benchmark functions having features that might be met in real-world problems, such as non-linearity and discontinuity [255]. This framework is based on a multidimensional tree, which is exploited to partition the search space and to select the best set of simple functions for each subspace, according to the characteristics underlying the considered subspace.

Results

The comparison of the performance of the metaheuristics exploited for the PE of 12 randomly generated RBMs, and for the optimization of a commonly used suite of benchmark functions, are presented here. Specifically, we employed the following functions from the CEC'17 benchmark problems for single-objective real-parameter numerical optimization

- the shifted/rotated Rosenbrock's (f_4) and Levy's (f_9) functions, as representatives of multimodal and non-separable problems characterized by a large number of local minima and whose optimum is not centered in **0**;
- the hybrid Ackley's (f_{13}), Griewank's (f_{15}), expanded Griewank plus Rosenbrock (f_{19}), and Schaffer's F7 (f_{20}) functions.

It is worth noting that hybrid functions are composed of multiple subcomponents that are assigned to different basis functions. The rationale is that hybrid functions represent strongly non-separable problems, with subcomponents that are characterized by specific behavior and sensitivity, supposedly mimicking real-world problems. Hybrid functions are designed to reproduce the peculiarities of real-world problems; however, we will show that these functions cannot capture all the characteristics of the fitness landscapes defined by the PE problem.

Table 5.1 summarizes the functioning settings of all the metaheuristics used in this work. The search space for the PE problem was set to $[10^{-12}, 100]^D$, while for the benchmark functions was set to $[-100, 100]^D$. The population size P was calculated by exploiting the following heuristic:

$$P = 32 \times \left\lceil \frac{\sqrt{D}}{2} \right\rceil,$$

which takes into account both the number of dimensions D and the Compute Unified Device Architecture (CUDA) warp size. The latter is critical to fully leverage the power of modern GPUs to simulate the dynamics of the RBMs by means of cupSODA. A population size $P = 96$ and $P = 128$ for $D = 25$ and $D = 50$, respectively, is obtained by applying the aforementioned heuristic.

We used the implementation of ABC provided by the SwarmPackagePy library (v. 1.0.0a5). CMA-ES, DE, EDA, and GA were implemented by using the Distributed Evolutionary Algorithms in `Python` (DEAP) framework (v. 1.0.2) [138]. We opted for this specific library because it simplifies the integration of cupSODA (v. 1.1.0) for the fitness evaluation. In the case of the algorithms implemented by using DEAP and SwarmPackagePy, we tested *off-the-shelf* optimization by using the default settings. For PSO, we used the most widespread settings relying on the analyses conducted in [448, 443, 409].

In order to collect statistically sound results, we performed 15 repetitions for each metaheuristic keeping track of the best fitness value found during each iteration, and then calculating the ABF. In the following figures, each metaheuristic is identified by a specific

Table 5.1 Functioning settings of the exploited optimization algorithms. n_0, n_e, n_s are the onlookers, employed and the scouts bees of ABC, respectively. χ_0^{ES}, σ_0^{ES} denote the initial mean and initial step-size of CMA-ES, respectively. *CR*, *F* represent the crossover rate and mutation strategy used for the DE/rand/1/bin strategy, respectively. χ_0^{EDA}, σ_0^{EDA}, λ μ are the centroid of the search space, standard deviation, new individuals and best individuals of EDA, respectively. p_{cr}, p_{mu}, $\kappa = 3$ represent the crossover rate (two point crossover), mutation rate (Gaussian mutation with $\sigma^{GA} = 1.0$), and number of individuals involved in each tournament in GA, respectively.

Algorithm	**Settings**
ABC	$n_o = \lfloor 0.5 \cdot n \rfloor, n_e = \lfloor 0.4 \cdot n \rfloor, n_s = P - n_o - n_e$
CMA-ES	$\chi_0^{ES} = \left(\beta_d^{max} + \beta_d^{min}\right)/2, \sigma_0^{ES} = 1.0$
DE	$CR = 0.25, F = 1.0$, mutation strategy: DE/rand/1/bin
EDA	$\chi_0^{EDA} = \left(\beta_d^{max} + \beta_d^{min}\right)/2, \sigma_0^{EDA} = 1.0, \lambda = n, \mu = 32$
GA	$p_{mu} = 0.2, p_{cr} = 0.99$, tournament selection ($\kappa = 3$), two point crossover, Gaussian mutation with $\sigma^{GA} = 1.0$
PSO	$c_{cog} = c_{soc} = 1.496$, $w = 0.729, v_d^{min} = 0, v_d^{max} = 0.2 \cdot \psi_d$, where $\psi_d = \beta_d^{max} - \beta_d^{min}$
FST-PSO	—

color: ABC is grey, CMA-ES is yellow, DE is pink, EDA is light blue, GA is blue, PSO is green, and FST-PSO is orange. We also tested an alternative version of FST-PSO whose minimum velocity throttling is disabled (rules $10-12$ [305]); we refer to this version as FST-PSO (no $\mathbf{v}_{min}$) and we depict it with the magenta color in the figures.

In Figures 5.17 and 5.18 we show the results concerning the convergence speed in terms of ABF obtained by the metaheuristics on the benchmark functions with $D = 25$ dimensions and on the RBMs with $D = 25$ kinetic constants to be estimated. The results reveal that in the case of the benchmark functions, CMA-ES is characterized by the best performances, except for the case of the f_9 and f_{20} functions, where the classic PSO achieves the best result. Interestingly, although CMA-ES largely outperformed the other algorithms in the case of the rotated/shifted Rosenbrock's function, it ranked last in the case of the hybrid Schaffer's F7 f_{20}. We also observe that the performances of DE are good with functions f_9 and f_{20}, while they are limited with the other benchmark functions. EDA was outperformed by the other algorithms on functions f_4, f_{13} and f_{19}, having an ABF orders of magnitude higher than the competitor methods. FST-PSO is characterized by mixed performances, yielding some high quality solution (e.g., f_4, f_{15}, f_{20}) but never outperforming the other algorithms in any of the tested benchmark functions.

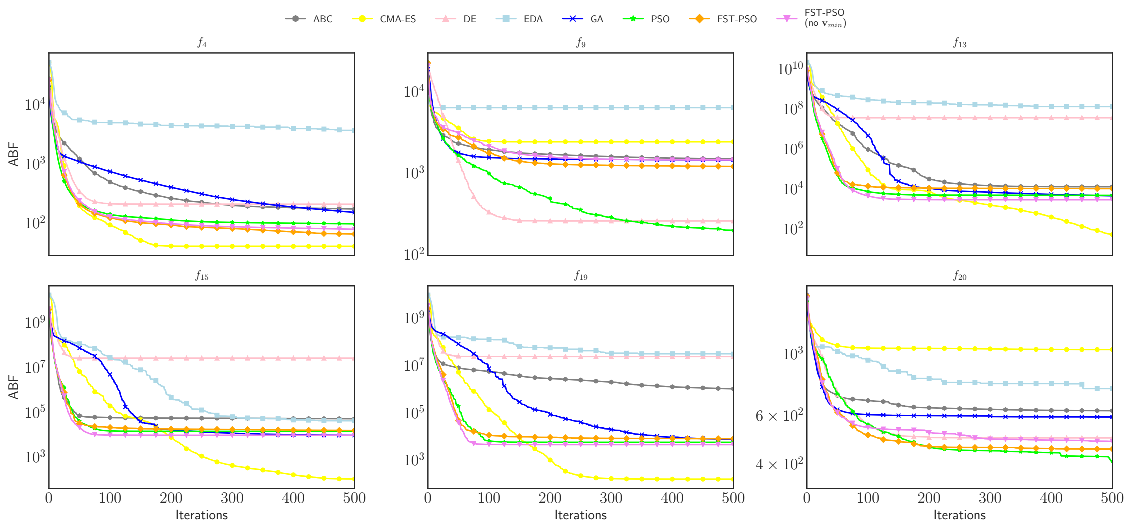


Fig. 5.17 Comparison of the performance in terms of ABF achieved by the metaheuristics on the benchmark functions with $D = 25$.

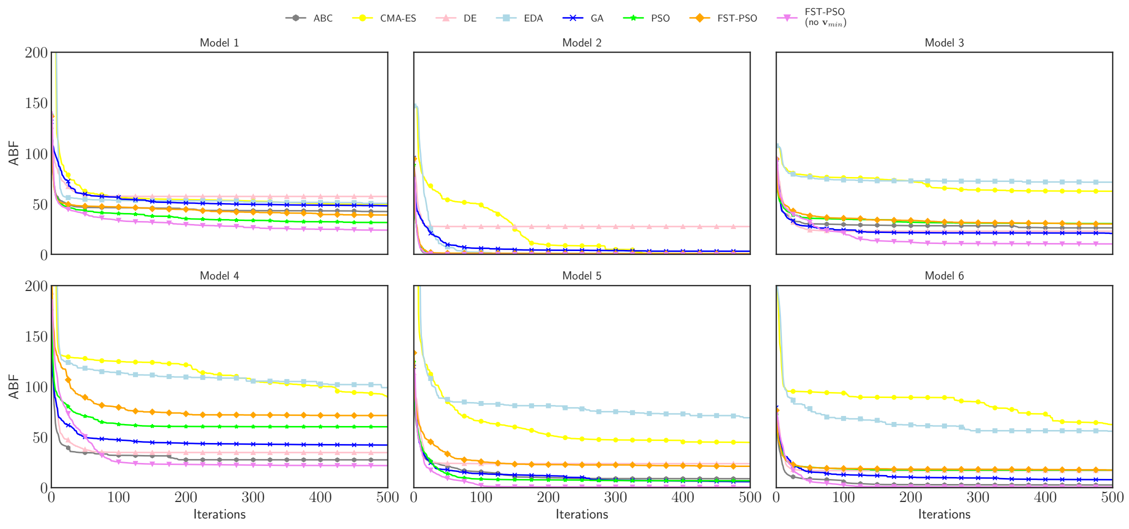


Fig. 5.18 Comparison of the performance in terms of ABF achieved by the metaheuristics for the PE of synthetic models characterized by 25 reactions and 25 molecular species.

Finally, both ABC and GA show average performances, always remaining in the middle of the ranking. Interestingly, ABC is characterized by a high variability in the results, as shown by the boxplots of the best solutions found across all runs (see Figures 5.21 and 5.27): the algorithm seems to yield either poor or very good solutions, probably according to the distribution of the bees at the beginning of the optimization phase. GA, however, seem to

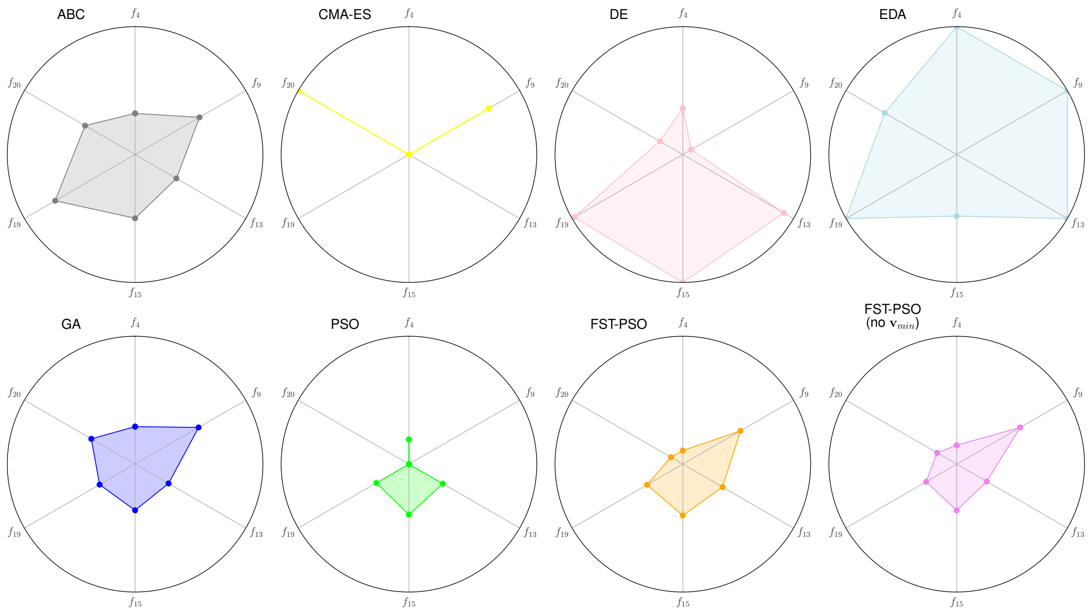


Fig. 5.19 Kiviat diagram showing the final ABF value obtained by the metaheuristics on the benchmark functions with $D = 25$.

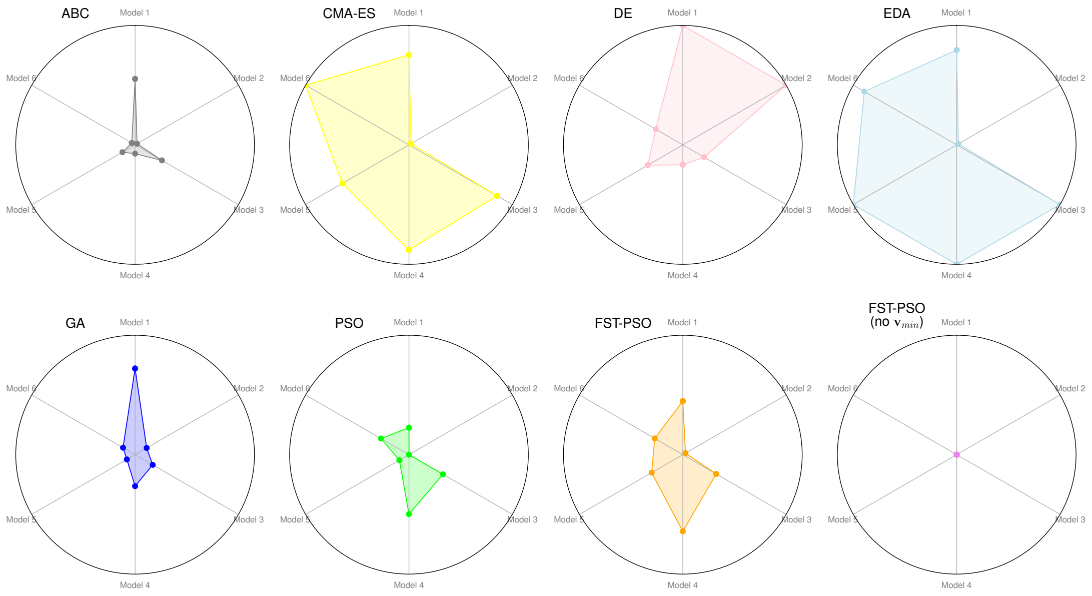


Fig. 5.20 Kiviat diagram showing the final ABF value obtained by the metaheuristics in the PE of synthetic models characterized by 25 reactions and 25 molecular species.

be characterized by a slower convergence (see, e.g., functions f_4, f_{13}, f_{15} and f_{19}) than the other metaheuristics.

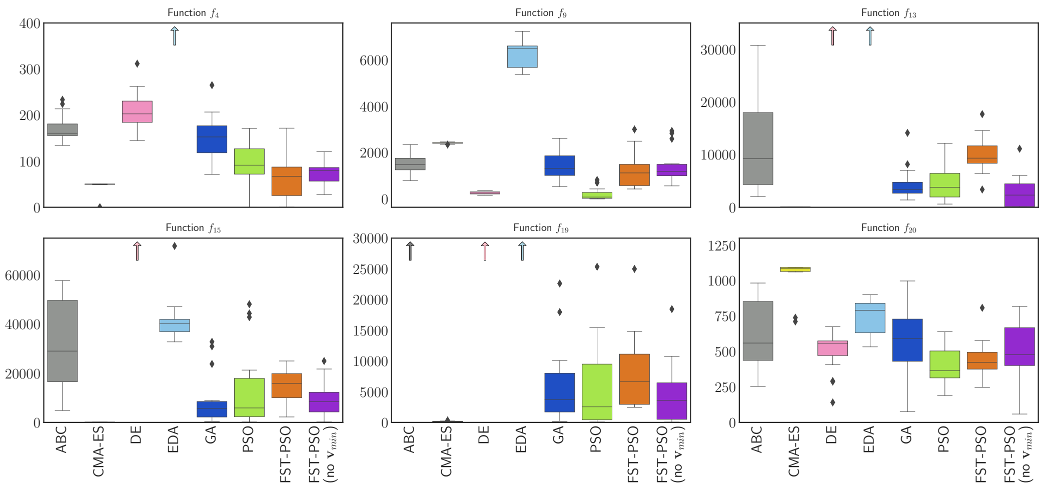


Fig. 5.21 Boxplots showing the distribution of the final best fitness values obtained by the metaheuristics on the benchmark functions with $M = 25$.

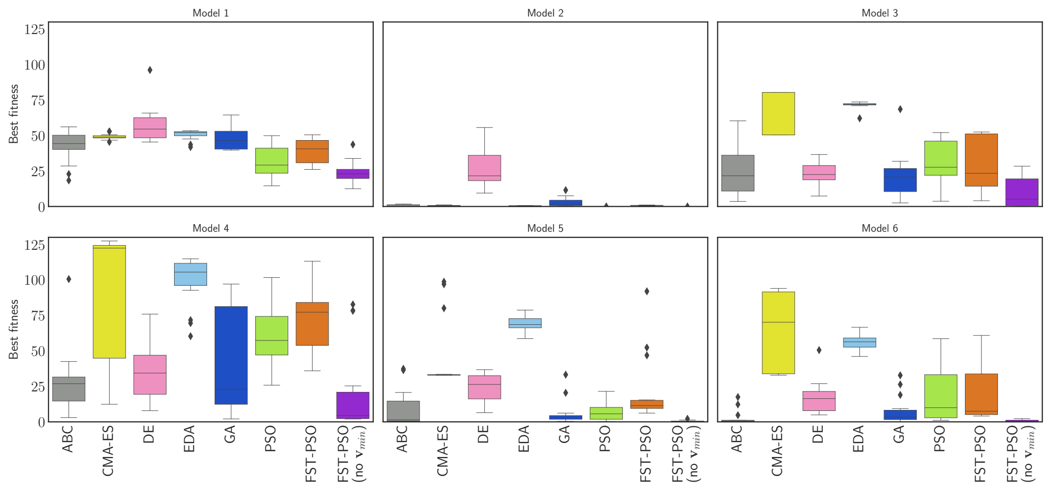


Fig. 5.22 Boxplots showing the distribution of the final best fitness value obtained by the metaheuristics in the PE of synthetic models characterized by 25 reactions and 25 molecular species.

Notice that the boxplots show the best fitting solutions found after 500 iterations. Since in some cases the algorithms could not converge properly the corresponding boxplot would be displayed outside the figure. We denote such situations using an arrow filled with the color corresponding to that algorithm (e.g., Figure 5.21, EDA in the case of function f_4).

When we consider the PE problem, the results turn out to be totally different, since the version of FST-PSO (no $\mathbf{v}^{min}$) consistently achieves the best results compared to the other metaheuristics. CMA-ES obtains a performance comparable to the other methods only

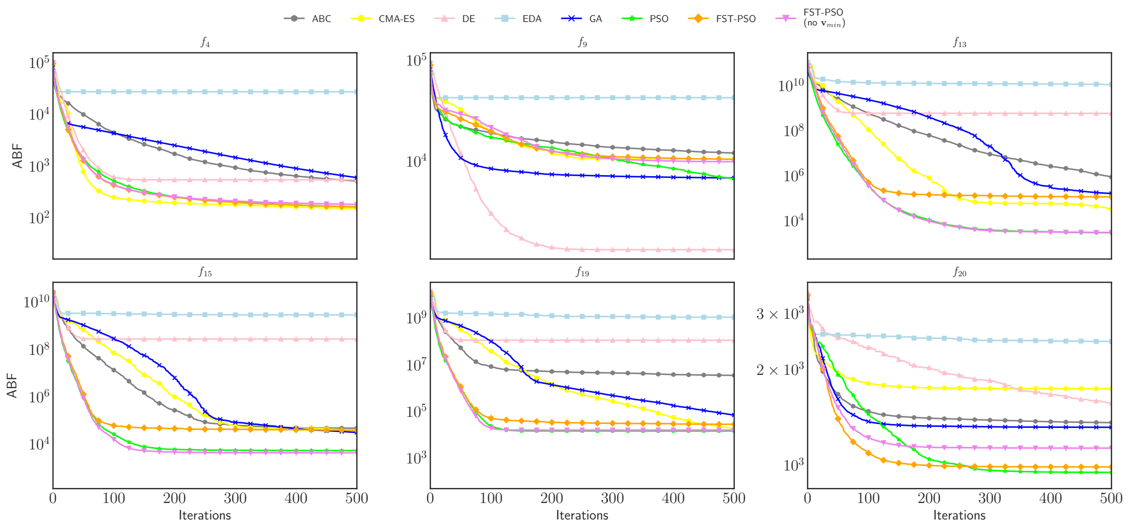


Fig. 5.23 Comparison of the performance in terms of ABF achieved by the metaheuristics on the benchmark functions with $D = 50$.

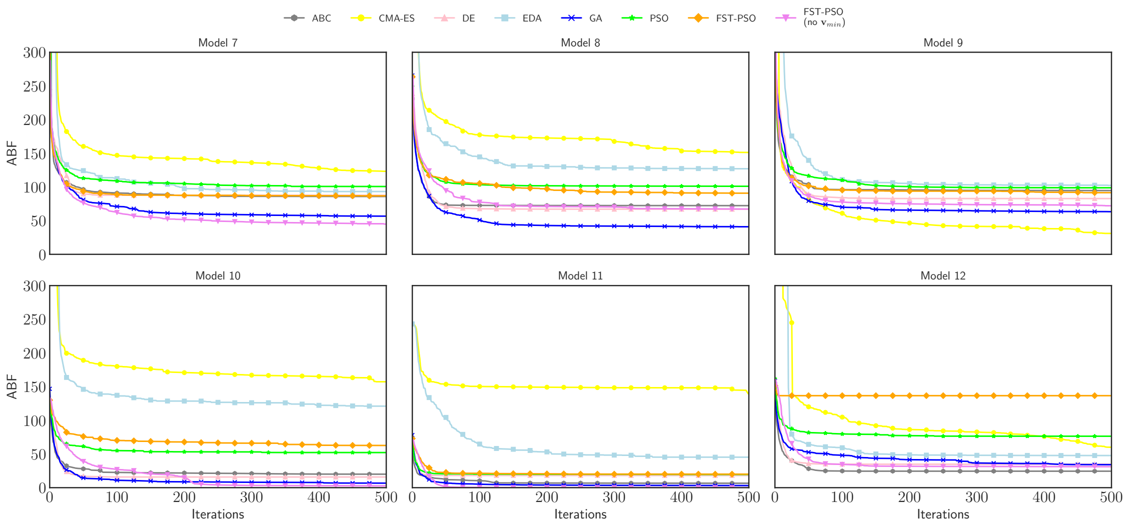


Fig. 5.24 Comparison of the performance in terms of ABF achieved by the metaheuristics for the PE of synthetic models characterized by 50 reactions and 50 molecular species.

in the case of Model 2 (where DE has the worst performance), and a worse performance (comparable to EDA) in all other cases. Interestingly, ABC tied FST-PSO performance in the case of Models 2 and 6 and, differently from the case of benchmark functions, it was not characterized by a high variance in the optimal solutions found (see Figures 5.22 and 5.28), confirming that the algorithm has a different behavior in the case of PE.

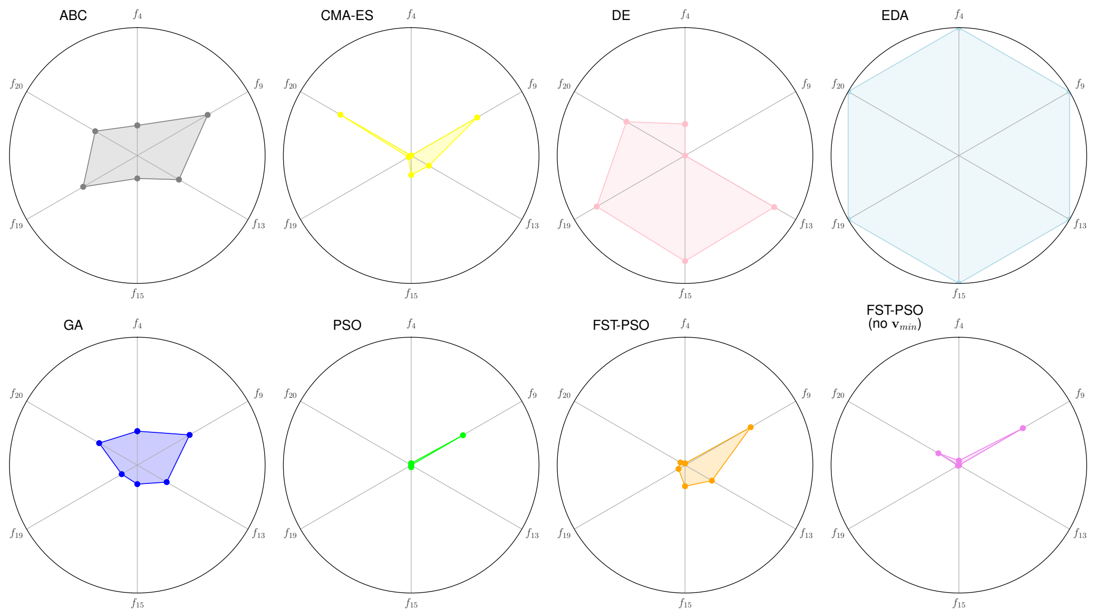


Fig. 5.25 Kiviat diagram showing the final ABF value obtained by the metaheuristics on the benchmark functions with $D = 50$.

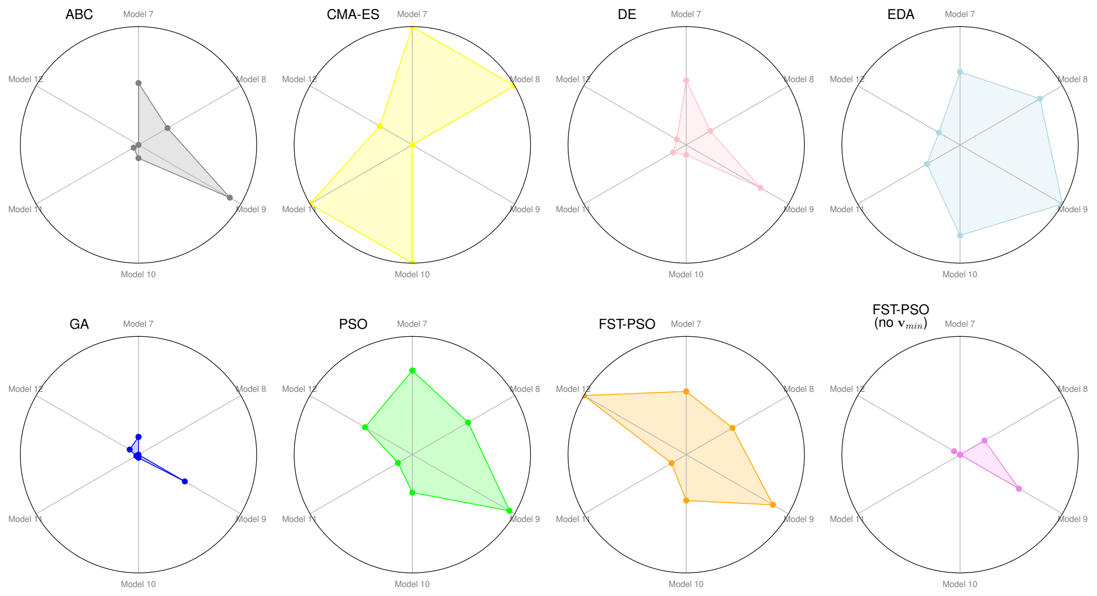


Fig. 5.26 Kiviat diagram showing the final ABF value obtained by the metaheuristics in the PE of synthetic models characterized by 50 reactions and 50 molecular species.

We executed further tests to analyze how the performance of the metaheuristics scale with the number of dimensions of the benchmark functions and the number of parameters to be estimated in the case of the PE problem. In Figures 5.23 and 5.24 we show the results

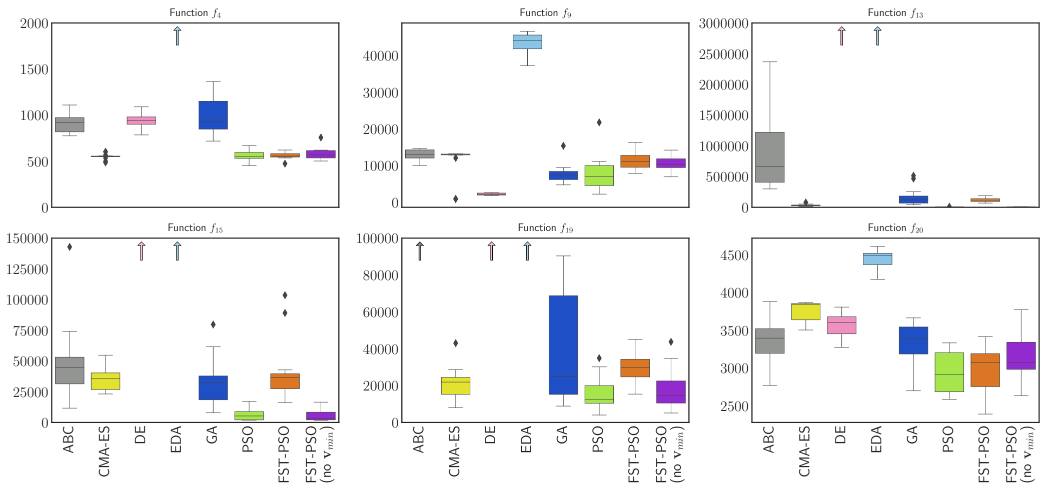


Fig. 5.27 Boxplots showing the distribution of the final best fitness values obtained by the metaheuristics on the benchmark functions with $M = 50$.

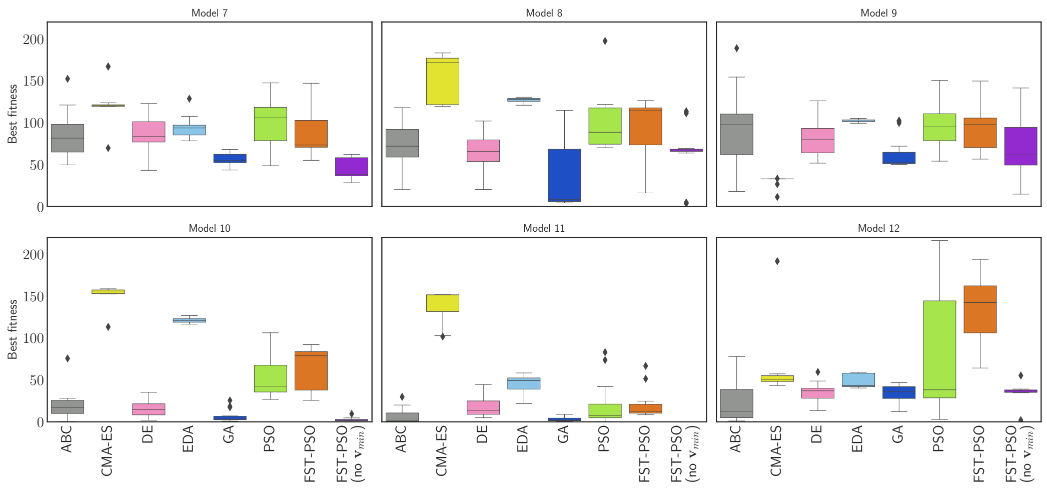


Fig. 5.28 Boxplots showing the distribution of the final best fitness value obtained by the metaheuristics in the PE of synthetic models characterized by 50 reactions and 50 molecular species.

obtained on the benchmark functions with $D = 50$ dimensions and on the RBMs with $D = 50$ kinetic constants to be estimated. Differently from the case of $D = 25$ dimensions, we observe that CMA-ES outperforms the other techniques only in the case of f_4. In the case of f_{19}, the final ABF of CMA-ES after 500 iterations is tied with PSO and FST-PSO, even though the convergence speed is far lower in the case of CMA-ES. Interestingly, the algorithm characterized by the best performances is the classic PSO except in the case of function f_9, contradicting what was observed in the case of $D = 25$. Moreover, taking into account benchmark functions with $D = 50$, EDA is no longer able to identify optimal solutions and seems to be extremely prone to premature convergence, while ABC has a considerably slower convergence speed with respect to the other metaheuristics. Again, when considering the PE problem, FST-PSO (no $\mathbf{v}^{min}$) consistently achieves the best results, outperforming the other methods and showing a higher convergence speed in the case of Model 7. CMA-ES consistently holds the worst performances, besides Models 9 and 12. GA seems to scale better, maintaining good or average performances over all 6 models.

An alternative representation of the results presented above is given in Figures 5.19, 5.20 and 5.25, 5.26 (for 25 and 50 dimensions of the benchmark functions and parameters to be estimated in the RBMs, respectively), in which we show the Kiviat diagrams obtained by plotting the ABF value achieved by the different metaheuristics during the last iteration of the optimization processes. Note that the lower the area described by the plot, the better the performance of the metaheuristic. Since the final fitness values are generally different among the benchmark functions as well as among the PE of different RBMs, we normalized these values in the range $[0,1]$. We observe how the performance of CMA-ES drastically decreases when applied to the PE problem, compared with the performance on the benchmark functions. The opposite holds for DE and GA, since these algorithms show better convergence properties in the case of PE with respect to benchmark functions. The performance of FST-PSO is also striking, especially when the fuzzy rules for minimum velocity are disabled. Note that this strategy leads to slightly worse results with the benchmark functions, but extremely good performances in the case of the PE, notably without the need for any functioning setting.

In order to investigate the existence of any statistical differences among the performances of the tested algorithms, we executed the Friedman's test [145] and the Bonferroni-Dunn's *post hoc* test [111]. Table 5.2 lists the ranks calculated using the ABF values achieved during the last iteration of all tests executed on the benchmark functions and the RBMs considered in this work. Since the p-values of the Friedman's test (reported in the table) allowed us to reject the null hypothesis (i.e., the difference in the performance of the algorithms is not statistically significant), we proceeded with the Bonferroni-Dunn's *post hoc* test to determine which algorithms are significantly better than the others. We thus calculated the critical differences (CDs) with 90% and 95% confidence levels, obtaining CDs equal to 0.895 and 0.982, respectively. Taking into account 95% confidence level, we formed groups of algorithms whose performances are not significantly different (denoted by Roman numerals in Table 5.2). ABC ranks in the third and fourth groups of algorithms in solving the benchmark functions with $M = 25$ and $M = 50$, respectively, while in the case of the PE problems it is capable of catching up with the second group of algorithms. PSO is characterized by an almost opposite trend; indeed, it ranks among the best algorithms (always

Table 5.2 Statistical comparison of the tested algorithms in solving the benchmark functions and the PE problem considering $M = 25$ and $M = 50$, calculated using the ABF values at the last iteration. The second row shows the p-values of the Friedman's test. Since the p-values allow us to reject the null hypothesis, we performed the Bonferroni-Dunn's *post hoc* test obtaining critical differences equal to 0.895 and 0.982 considering 90% and 95% confidence levels, respectively. For each column, the results are expressed as: ranking – group (obtained with 95% confidence level and denoted with Roman numerals). Note that an algorithm can belong to more than one group.

	$M = 25$		$M = 50$	
	Benchmarks	PE	Benchmarks	PE
	p-value=0.0014	p-value=0.00025	p-value=0.00020	p-value=0.00014
ABC	6.000 – III	3.167 – II	5.833 – IV	3.500 – II
CMA-ES	3.000 – I, II	6.333 – IV	4.000 – III	6.500 – IV, V
DE	5.833 – III	5.333 – III	5.667 – IV	3.667 – II
EDA	7.833 – IV	7.500 – V	8.000 – V	7.167 – V
GA	3.833 – II	3.833 – II	4.500 – III	2.000 – I
PSO	2.833 – I	3.833 – II	1.667 – I	5.833 – III, IV
FST-PSO	3.667 – I, II	5.000 – III	3.667 – III	5.500 – III
FST-PSO (no $\mathbf{v}_{min}$)	3.000 – I, II	1.000 – I	2.667 – II	1.833 – I

in the first group) when solving benchmark functions (resulting the best choice in the case of $M = 50$), whereas it belongs to the second group regarding the PE problem when $M = 25$, and in the third and fourth groups when $M = 50$, showing how its performance decreases while the number of dimensions increases. The results obtained by FST-PSO are strictly comparable with those achieved by PSO, even if it generally performs better (worse) in the PE problem (benchmark functions). Moreover, FST-PSO generally outperforms DE in all cases except for the PE with $M = 50$. Disabling the fuzzy rules for the minimum velocity throttling, the results are quite different. As a matter of fact, FST-PSO (no $\mathbf{v}_{min}$) ranks always among the best algorithms, taking into account both benchmark functions and the PE problem. GAs obtain the best results when the number of dimensions increases, becoming highly competitive in solving the PE problem, ranking first together with FST-PSO (no $\mathbf{v}_{min}$) when $M = 50$. It is also competitive in solving benchmark functions, placing in the second and third groups. In this case, the performance of GAs decreases as the number of dimensions increases. CMA-ES shows quite good performance with benchmark functions, being always in the first three groups of algorithms, while it is not competitive in the case of the PE problem. Finally, the achieved results highlight that EDA obtains the worst performance considering both the benchmark functions and the PE problem, attaining the last group in all tests.

To summarize, the analysis conducted on the benchmark functions highlighted that the best algorithms are PSO, FST-PSO, FST-PSO (no $\mathbf{v}_{min}$) and CMA-ES. Among them, PSO might be employed due to its simplicity; however, its performance are strongly related to the values of its settings. FST-PSO and its variant FST-PSO (no $\mathbf{v}_{min}$) can overcome this limitation, thus resulting the most suitable algorithms to deal with benchmark functions. Regarding the PE problem, FST-PSO (no $\mathbf{v}_{min}$) is generally capable of outperforming the other tested algorithms, being the best metaheuristic when $M = 25$ and ranking first together with GAs when $M = 50$.

Since both CMA-ES and EDA exploit normal distributions to generate new individuals, their performances could be affected by the peculiar logarithmic distribution of kinetic parameters. We point out that alternative semantics for the parameters can radically change the performances of the metaheuristics. Specifically, we show that a simple logarithmic transformation of the parameters can turn the previously outperformed algorithms into competitive alternatives. In order to investigate this conjecture, we modified CMA-ES, EDA and FST-PSO to change the semantics of the parameters to a logarithmic scale. To be more precise, the putative parameters were bounded in the interval $(0, 1)$ and each value k_i was converted to the actual kinetic parameter k_i'—used for the fitness evaluation—by means of the following transformation:

$$\begin{aligned} \phi_i &= \log_{10}(\beta_i^{max}) + \Big(\log_{10}(\beta_i^{min}) - \log_{10}(\beta_i^{max})\Big)\, k_i, \\ k_i' &= 10^{\phi_i}. \end{aligned} \qquad (5.3)$$

We denote by CMA-ES-log, EDA-log and FST-PSO-log the three modified algorithms. We show in Figure 5.29 a comparison of the performances of the three modified algorithms (solid lines) with respect to the original methods (dashed lines). The test was carried out on Model 10, in which both CMA-ES and EDA showed, by far, the worst performances.

According to our results, thanks to the transformation in Equation 5.3, the performance of CMA-ES-log is radically different from classic CMA-ES, with a final ABF very close to zero and an extremely quick convergence. The performance of EDA-log (whose σ_0^{EDA} was set to 0.1 because of the modified search space) strongly improved with respect to classic EDA; however, it was repeatedly unable to converge to an optimal solution, keeping the final ABF above 50. Even though CMA-ES-log was able to rapidly converge, the result achieved by FST-PSO-log is even better and highlights how the logarithmic semantics can help *all* algorithms for the PE problem. This circumstance further reveals that benchmark functions cannot capture the intrinsic complexity of biochemical PE.

Overall, the results achieved in this work point out that the performance of the metaheuristics can drastically change according to the context of application, showing that the fitness landscapes identified by classic benchmark functions are completely different from those characterizing the PE problem. We argue that a novel set of benchmark functions,

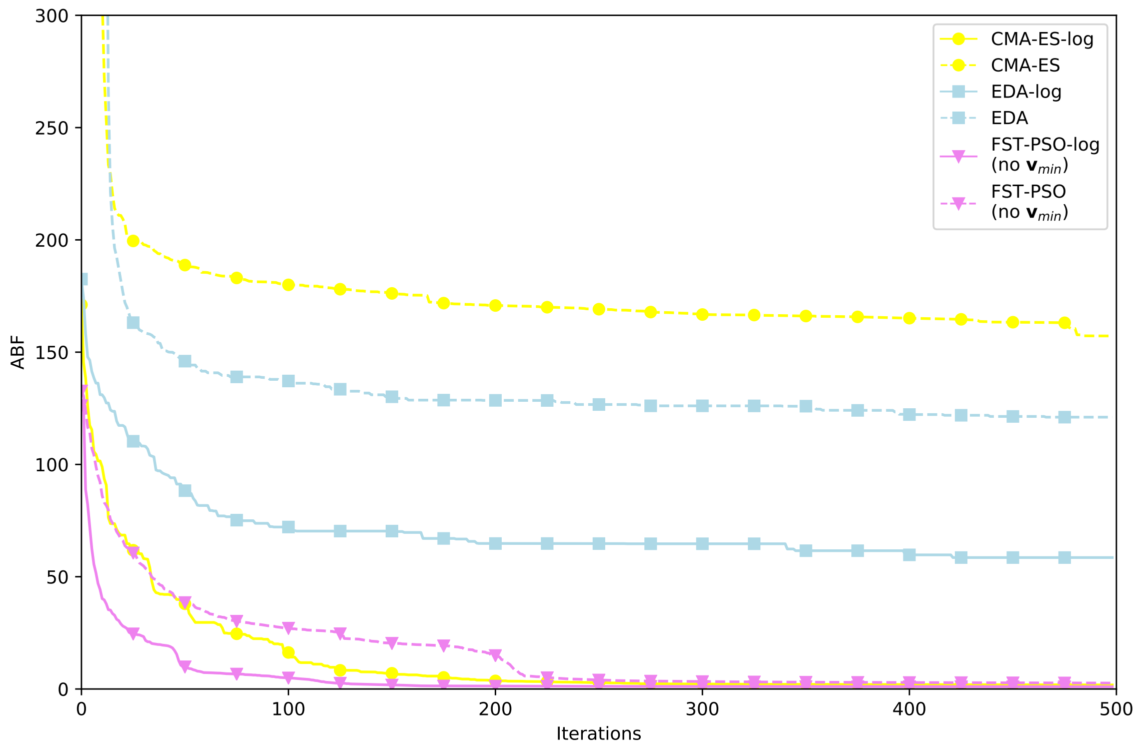


Fig. 5.29 Comparison of the performances of CMA-ES, EDA and FST-PSO with normal and logarithmic semantics of parameters.

designed to mimic the characteristics of real-world problems, is necessary to achieve a better understanding and a thorough evaluation of the performance of the metaheuristics. These benchmark functions should be defined attempting to resemble the fitness landscapes of a variety of real-world problems. Although some preliminary efforts were devoted to create functions similar to the PE problem [73], we are still far from a complete and reliable reproduction of its intrinsic characteristics. In principle, real-world problems should be applied for benchmarking, since they provide a valuable contribution to experimental research practice [141]. Differently from benchmark functions, the structural features underlying real-world optimization problems are often not well characterized [261]; thus, additional research must be performed to understand how novel benchmark functions could be designed to replicate their peculiarities. As a matter of fact, defining benchmark functions inspiblue by real-world problems is not trivial, since it requires the preliminary design and development of novel *ad hoc* methods to analyze and classify optimization problems, as well as automatic methods (by using, e.g., Genetic Programming [246] or hierarchical fitness assignment methods based on statistical tests [267, 268]) to devise arbitrary functions characterized by analogous fitness-space features.

The results of our tests highlighted that CMA-ES is one of the best choices for the optimization of benchmark functions, but its performance turned out to be worse than most of the other metaheuristics when applied to the PE of biochemical systems in 10 out of 12 RBMs. Since both CMA-ES and EDA exploit normal distributions to generate new individuals, their performances are probably affected by the peculiar log-uniform distribution of kinetic parameters [409]. We empirically proved this conjecture by repeating the PE tasks using a logarithmic semantics for the putative parameters, showing that all algorithms benefit from this solution and, in particular, CMA-ES was now able to efficiently converge to high quality solutions. We will further investigate the logarithmic exploration of the parameter space, a topic that we previously tackled by considering also the population initialization [73] and particle reboot [409] in PSO. We argue that the performance of some algorithms in specific real-world problems can be strongly improved by transforming, or adapting, the representation of the solutions. Although it was possible for us to define an effective transformation in the case of PE, this task is generally not straightforward to perform. In particular, we speculate that the automatic design of the optimal transformation for any problem might be as difficult as solving the optimization problem itself. Due to its relevance in the context of optimization problems, we plan to investigate this topic in the near future.

Finally, we observed that the version of FST-PSO where fuzzy rules for the minimum velocity throttling are disabled (i.e., not leveraging turbulence [2]) appears to be the best choice for PE, although its convergence speed in the case of the benchmark functions is worse than classic PSO. Anyway, PSO requires the selection of multiple functioning settings, which is not necessary in the case of FST-PSO. As a further extension of this work, we will define improved alternative fuzzy rules (or approaches) to automatically set the minimum velocity, in order to define a completely multi-purpose methodology effective both in the case of benchmark functions and real-world problems.

5.3 Multi-swarm PE of small-scale models

The starting point of the work presented in this section is the multi-swarm version of PSO proposed in [304, 303], an efficient optimization method that takes into account as target of the PE a set of DTTS, obtained under different experimental conditions, the typical scenario of biological laboratory research. Multi-swarm approaches, which belong to distributed island-based algorithms [8], partition the population into a set of sub-populations, called islands, executing independent parts of the whole optimization process. Occasionally, information is transferred among the islands, aiming at introducing more diversity into the sub-populations, thus preventing early convergence to local optima. In the multi-swarm PSO, a single series of target data is assigned to each swarm and, during the optimization, the best particle of each swarm can migrate to the aim of obtaining a set of reaction constants that simultaneously allow for fitting all the experimental data.

Here we present an efficient implementation of such optimization methodology, called MS^{2}PSO, which allows for drastically accelerating the computation by exploiting a numerical integrator of ODEs accelerated on GPUs and the Message Passing Interface (MPI) to leverage multi-core CPUs. This work is motivated by the fact that the computation time required to run a PE increases both with the number of swarms (and particles), and with the complexity of the mathematical model under investigation.

In particular, since the optimization process requires the calculation of a massive number of fitness values, MS^{2}PSO relies on cupSODA. In addition, to leverage the power of the modern computer clusters, we introduce a further level of parallelism by exploiting the Master-Slave distributed programming paradigm that allows for offloading all the calculations required by MS^{2}PSO[119] onto multiple GPUs (when available) and multi-core CPUs. In MS^{2}PSO, the Master orchestrates the communication among the Slave processes, which in turn execute the PE of each PSO swarm involved in the optimization process.

Parallel and distributed computing applications have been widely adopted in Computational Intelligence [9, 164]. Traditional Master-Slave architectures represent a valuable solution in both Evolutionary Computation [119, 209] and Swarm Intelligence [206]. Yang *et al.* proposed a Master-Slave PSO algorithm, employing a Master swarm for exploitation and a Slave swarm for exploration [479]. In the field of Systems Biology, such a kind of HPC paradigm has been recently used to accelerate the PE of large-scale models [336].

5.3.1 Master-Slave approach

The multi-swarm version of PSO employed in MS^{2}PSO is structured as follows. For each e-th initial condition, with $e = 1, \dots, E$, we consider a swarm σ_e, consisting of P particles, in which the i-th particle can be identified by the position vector $\mathbf{x}_i^e$ that codifies a model parameterization $\mathbf{k}_e = (k_{e,1}, \dots, k_{e,D})$, whose components correspond to the values of the kinetic constants of the reactions in $\mathcal{R}'$.

Each vector $\mathbf{k}_e$ associated with a particle is used to execute a simulation with cupSODA in order to generate, for each target species S_q, the sets of concentrations $X_q^{\mathbf{k}_e}(\tau_f)$. The fitness of the particle is then evaluated according to Equation 5.2, using the experimental data $Y_q^e(t_f)$ corresponding to the e-th experimental condition. By so doing, each swarm performs the estimation of kinetic constants independently from the other swarms and determines, for each iteration, its global best particle whose position vector represents the best parameterization $\mathbf{k}_e^{best}$ that overlaps the DTTS under the e-th condition in the best possible way. On the contrary,

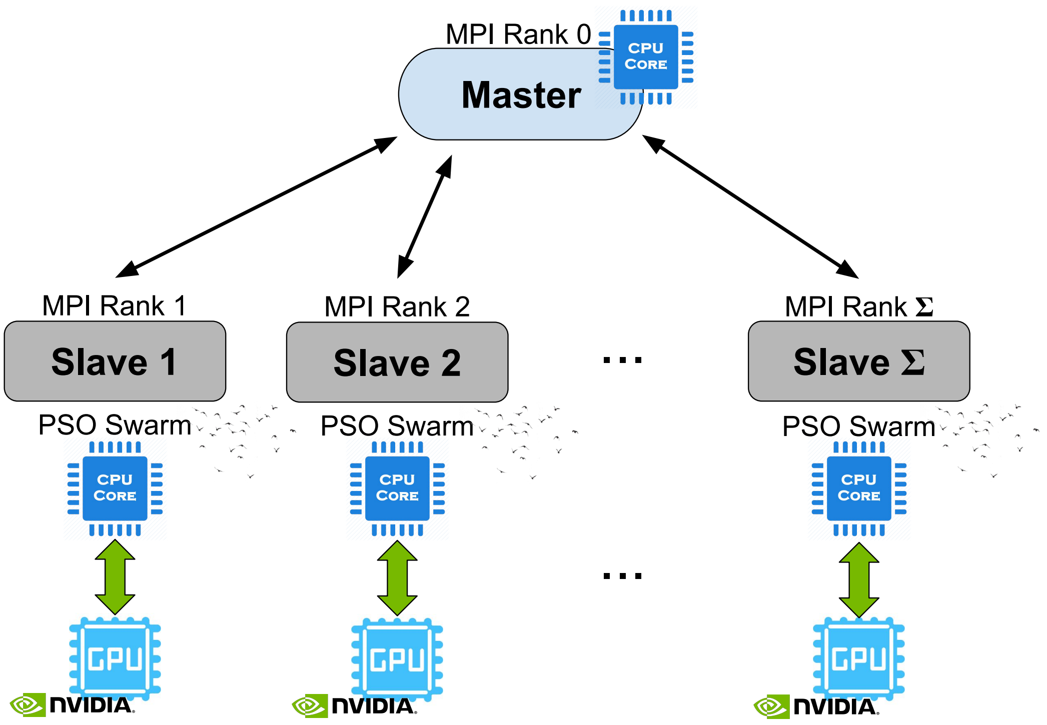


Fig. 5.30 Scheme of MS^{2}PSO: the Master process orchestrates all the Slaves assigning one or more PSO swarms to each Slave, which then run the PE task exploiting cupSODA to solve in parallel all the required deterministic simulations.

during each generation, the swarm also identifies the worst particle $\mathbf{k}_e^{worst}$, corresponding to the parameterization that leads to the worst overlap under the e-th condition.

Afterwards, in order to estimate a set of kinetic constants that is common to all swarms, and that is able to reproduce the expected system dynamics under all experimental conditions, we let particles migrate among swarms. The migration takes place at regular intervals, every $\kappa IT_{\mathtt{mig}}$ iterations, with $1 \leq \kappa \leq \lfloor IT_{\mathtt{MAX}}/IT_{\mathtt{mig}} \rfloor$ and $1 \leq IT_{\mathtt{mig}} \leq IT_{\mathtt{MAX}}$, where $IT_{\mathtt{MAX}}$ is the maximum number of iterations of the PSO. To better explain migration, we formalize the topology of MS^{2}PSO as a directed graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{A} \rangle$, where $\mathcal{V} = \{\sigma_1, \dots, \sigma_E\}$ is the set of E vertices (the swarms) and $\mathcal{A} = \{(\sigma_{e'}, \sigma_{e''}) \mid \text{particles can migrate from swarm } \sigma_{e'} \text{ to swarm } \sigma_{e''}\}$ is the set of edges. We also assume that, considering any edge in $\mathcal{A}$, exactly one particle migrates from $\sigma_{e'}$ to $\sigma_{e''}$, to assure that the size P of each swarm is not altered by migration.

Different interconnection topologies among the swarms can be defined, for instance the one proposed in [304] or [366]. Here we consider a ring interconnection topology among swarms; then, the migration acts as follows: for each edge $(\sigma_{e'}, \sigma_{e''})$ in $\mathcal{A}$, at the $IT_{\mathtt{mig}}$-th iteration the worst particle $\mathbf{k}_{e''}^{worst}$ of swarm $\sigma_{e''}$ is removed and replaced by the global best particle $\mathbf{k}_{e'}^{best}$ of swarm $\sigma_{e'}$.

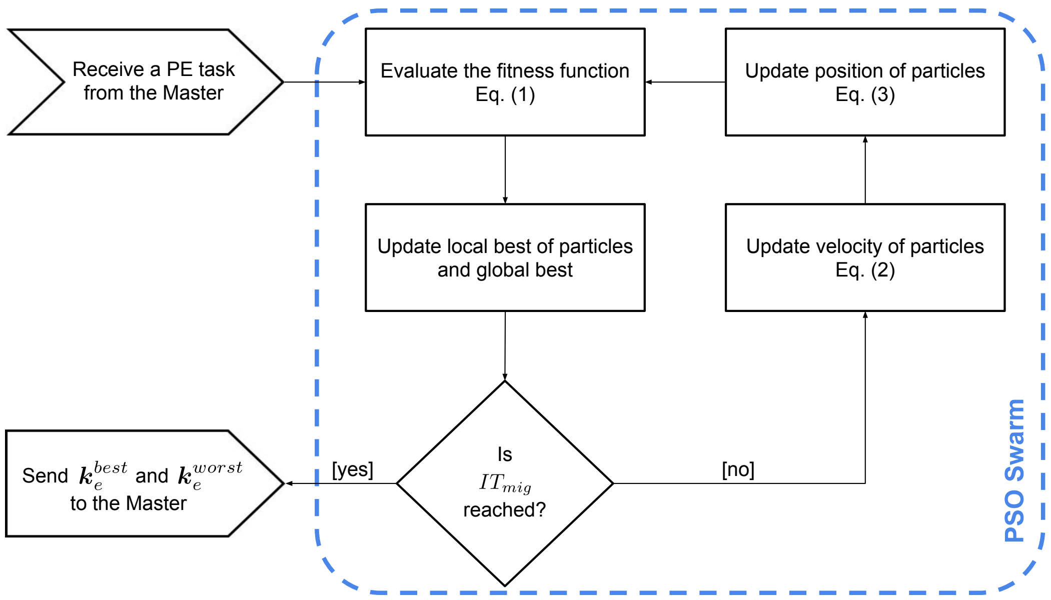

Receive a PE task from the Master
Evaluate the fitness function Eq. (1)
Update position of particles Eq. (3)
Update local best of particles and global best
Update velocity of particles Eq. (2)
Is IT_{mig} reached?
[yes]
[no]
Send $\boldsymbol{k}_e^{best}$ and $\boldsymbol{k}_e^{worst}$ to the Master
PSO Swarm

Fig. 5.31 Workflow of a single PSO swarm execution. The Master assigns to the Slaves a different PE task. Each Slave σ_e performs $IT_{\mathtt{mig}}$ iterations and communicates both $\mathbf{k}_e^{best}$ and $\mathbf{k}_e^{worst}$ to the Master. This process is repeated until the maximum number of iterations $IT_{\mathtt{MAX}}$ is reached by all swarms.

Master-Slave approaches in Computational Intelligence generally rely on parallel fitness evaluations, since each individual can be evaluated independently. More specifically, there is a process, called Master, which orchestrates the communication among the other processes called Slaves. In the case of MS^{2}PSO, the Master efficiently manages the Slaves and the communication among them to solve the PE tasks, as schematized in Figure 5.30. Please notice that in our implementation, both the Master and the Slave processes can run on different cores of the same CPU. Figure 5.31 shows the workflow of a single PSO swarm execution.

We can summarize the functioning of MS^{2}PSO as follows:

1. the Master allocates the resources and offloads the experimental data, corresponding to the E swarms, onto the available Σ Slaves (in general, $\Sigma \leq E$). During this phase, each Slave generates the initial population of swarm σ_e;

2. each Slave σ_e (with $e = 1, \ldots, E$) executes the assigned PE task, running $IT_{\mathtt{mig}}$ iterations of PSO, independently of the other Slaves. In particular, at each iteration the Slaves execute in a parallel fashion on the GPU the deterministic simulations required to calculate the fitness values of all particles in the swarm, by exploiting cupSODA;

3. after $IT_{\mathtt{mig}}$ iterations, each Slave σ_e communicates both $\mathbf{k}_e^{best}$ and $\mathbf{k}_e^{worst}$ to the Master;

4. as soon as all Slaves have communicated the best and worst particles found, the Master executes the migration, by using a ring topology among swarms, following the procedure described above;

5. the process is iterated from *Step* 2 until the maximum number of iterations $IT_{\texttt{MAX}}$ is reached by all swarms;

6. the Master performs a sorting of the $\mathbf{k}_e^{best}$ particles, and returns the particle with the lowest fitness as final result.

Note that when $\Sigma < E$ GPUs are available, the Master process must assign more than one swarm to the Slaves, whose execution is then scheduled sequentially. Moreover, if required, alternative fitness functions might be implemented on Slave nodes by fetching and processing on the CPUs the raw dynamics received from cupSODA. This scenario is technically feasible, even though more computationally demanding.

The proposed Master-Slave approach was entirely developed with the `Python` programming language (version 2.7.12) exploiting `mpi4py`, which provides bindings of the MPI specifications for `Python` to leverage multi-core architectures [98].

5.3.2 Results

The performance obtained by MS^{2}PSO, in terms of running time required to execute the PE, is evaluated under different configurations in which different numbers of GPUs as well as different numbers of cores of the CPU are exploited.

For what concerns the settings used in the multi-swarm PSO, in all tests, we have: (*i*) damping boundary conditions; (*ii*) maximum velocity $v_d^{max} = 0.2 \cdot |\beta_d^{max} - \beta_d^{min}|$, for all $d = 1, \dots, D$; (*iii*) number of iterations $IT_{\texttt{MAX}} = 400$. (*iv*) migration interval $IT_{\texttt{mig}} = 40$ iterations; (*v*) ring topology for migration. We also set $c_{cog} = c_{soc} = 1.494$ and we kept the inertia factor constant during the optimization (i.e., $w = 0.729$), according to the results presented in the previous sections. Finally, each swarm has the same size P, determined by the following heuristic that considers successive multiples of 32, increasing with the search space dimensions D:

$$P = 32 \times \left(1 + \left\lfloor \frac{D}{10} \right\rfloor\right). \tag{5.4}$$

The principle underlying Equation 5.4 is to create a population proportional to both the complexity of the problem and the size of a CUDA warp, in order to fully exploit the acceleration given by cupSODA.

The performance of MS^{2}PSO, in terms of computation time required to solve the PE problem, are evaluated by performing 15 independent repetitions of the PE of three realistic *in silico* generated biochemical models with 10, 20 and 30 chemical reactions and molecular species. These models are obtained by relying on the tool described in Section 4.1, which is capable of automatically yielding realistic networks of reactions with different size and features.

In particular, each model is generated considering the following features:

- the initial concentrations $\mathbf{x}$ at time $t = 0$ are sampled from the uniform distribution $[10^{-6}, 1)$;
- the kinetic constants $\mathbf{k}_e$, concerning the e-th experimental condition, with $e = 1, \ldots, E$, associated with the reactions are sampled from the logarithmic distribution in $[10^{-8}, 10]$;
- the final network of reactions must consist in a single connected component;
- only zero, first, or second order reactions are allowed;
- reactions can have zero, one or two products;
- we assume $E = 4$ distinct experimental conditions for each model, by perturbing the initial concentration of some chemical species.

The fitness value of the individuals is calculated by considering a DTTS having a number of species $H = \lceil 0.15 \cdot N \rceil$. Therefore, in the following tests we considered $H = 2$, $H = 3$ and $H = 5$ for the models with 10, 20 and 30 reactions and species, respectively. Finally, according to Equation 5.4, we used a swarm size of $P = 64$, $P = 96$ and $P = 128$ for $D = 10$, $D = 20$ and $D = 30$, respectively.

We first assess the accuracy of the multi-swarm PSO, which is at the basis of MS^{2}PSO. For each model, we verified if the best set of kinetic parameters, found at the end of the PE, allows for obtaining simulated dynamics that overlap the DTTS in each of the $E = 4$ initial conditions. As an example, we show in Figure 5.32 the results obtained in the case of the model consisting in 20 chemical reactions and molecular species, where the simulated dynamics (solid lines) perfectly overlap the DTTS (red dots) in all of the considered initial conditions and for each species.

The performance of MS^{2}PSO are then evaluated by executing 15 repetitions of the PE of the three analyzed synthetic models using four computational platforms equipped with different CPUs and Nvidia GeForce GPUs: GTX Titan Z, GTX Titan X, GTX 960M, and GTX 1050. The technical specifications of these platforms are summarized in Table 5.3. We defined specific tests to investigate the capability of MS^{2}PSO of leveraging both single- and

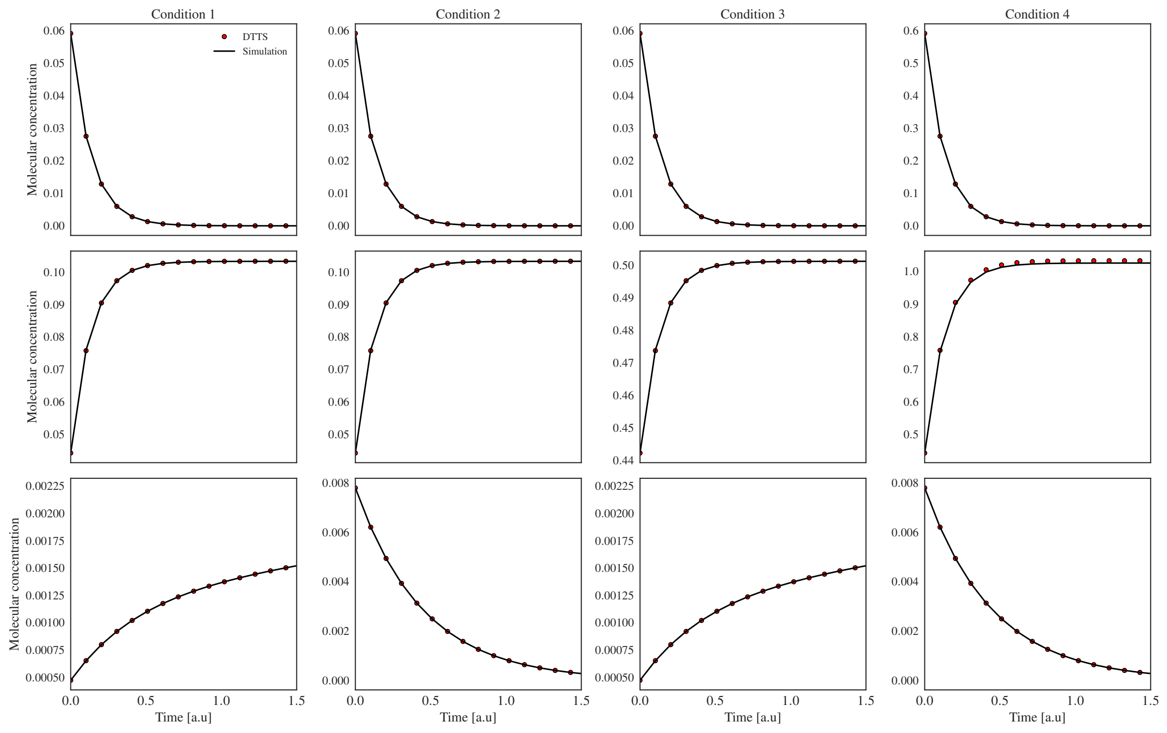


Fig. 5.32 Dynamics of the synthetic model characterized by 20 chemical reactions and 20 molecular species, obtained using the kinetic constants of the best solution generated by the four swarms of PSO (solid lines) compared to the DTTS (red dots), under different initial conditions.

multi-GPU architectures (when available) together with multi-core CPUs. Indeed, among the considered computing platforms, #3 and #4 are multi-GPU systems: the first is equipped with a dual-GPU GTX Titan Z, while the second is a cluster node equipped with 4 GTX Titan X.

Figure 5.33 shows the average running times (top) and the boxplots (bottom) achieved by exploiting the 4 computational platforms. If we compare the results obtained when considering the single-GPU configurations, in which the execution of the PE of the 4 swarms is scheduled sequentially by the Master process on the GPU, we can observe that for both models with 10 and 20 parameters to be estimated, the performances are comparable in the case of platforms #1, #2 and #3, while platform #4 has, in general, higher running times (see Figure 5.33, top). We believe that, despite the clock frequency of the GTX Titan X (platform #4) is higher than that of the GTX Titan Z (platform #3), the performance of the former platform are impaired by the CPU (whose clock frequency is lower than the others). In the case of the model with 30 parameters to be estimated, the efficiency of the GPU is the key factor influencing the overall performance of MS^{2}PSO; as a matter of fact, the GTX 1050,

Table 5.3 Hardware and software characteristics of the computing platforms used to evaluate the performance of MS^{2}PSO.

Platform	**GPU**	**Cores**	**Clock** [MHz]	**Mem. BW** [GB/s]	**CPU**	**Cores**	**Clock** [GHz]	**Operating System**
#1	GTX 960M	640	1176	80	Intel Core 6700HQ	4	3.50	Ubuntu 14.04
#2	GTX 1050	640	1455	112	Intel Core 7700HQ	4	3.80	MS Windows 10
#3	GTX Titan Z	2×2880	876	672	Intel Core i7-4790K	4	4.40	Ubuntu 16.04
#4	GTX Titan X	3072	1075	336.5	Intel Xeon E5-2620 v3	6	3.20	CentOS 6.9

characterized by the highest clock frequency, allows for obtaining the lowest average running times.

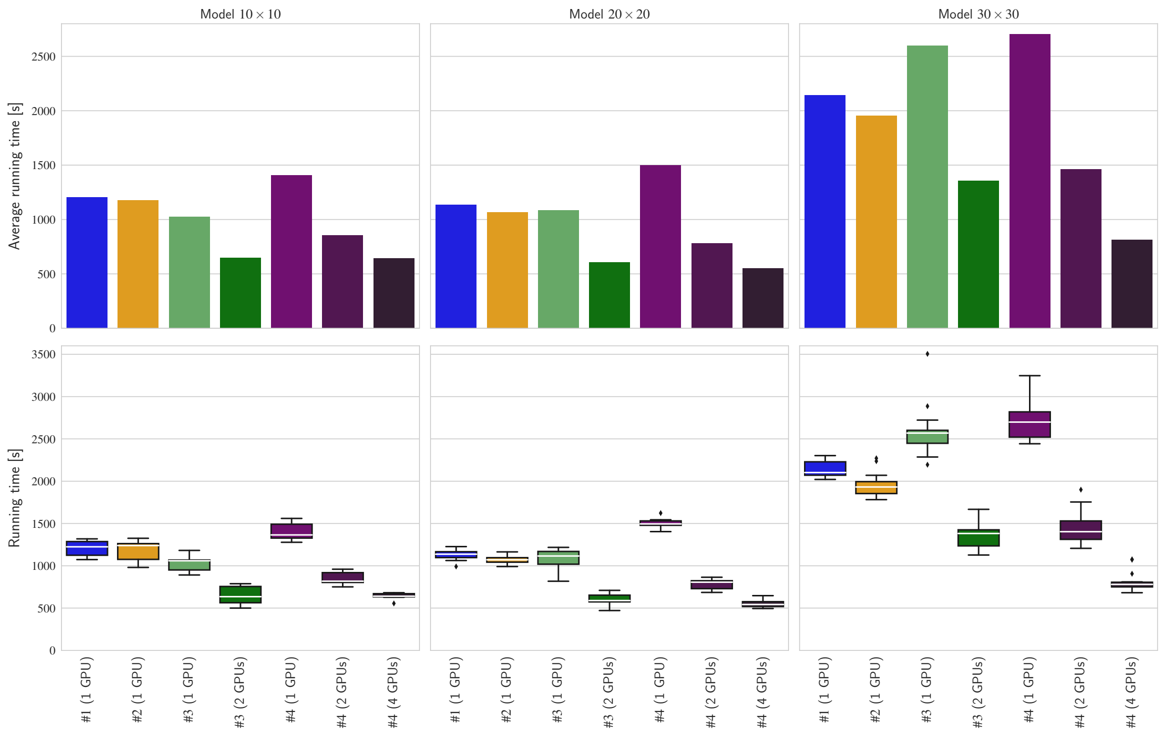


Fig. 5.33 Running time for 15 independent repetitions of MS^{2}PSO for the PE of the analyzed synthetic biochemical models with 10 (left), 20 (center), and 30 (right) parameters to be estimated. *Top panel:* bar diagram depicting the average running time obtained by each platform. *Bottom panel:* boxplots showing the distribution of the running times of each platform (the median is denoted by a horizontal white line, whisker value is 1.5 and outliers are displayed as black diamonds).

Considering the boxplots reported in Figure 5.33 (bottom), we can observe that in the case of the models with 10 and 20 reactions and species, the capabilities of the different GPUs do not strongly emerge, since the simulation of these relatively small models do not represent the most computational intensive task of the PE, thus the overall process is CPU-bound. This is confirmed by the fact that platform #3, which is equipped with the CPU characterized by the highest clock frequency, obtained the lowest running time. Moreover, the boxplots show,

Table 5.4 Average speed-up achieved by platforms #3 and #4 with multi-GPU configurations with respect to the corresponding single-GPU configurations.

Platform	**Model size**	**Average speed-up**	
		2 GPUs	**4 GPUs**
#3	10×10	1.6×	N/A
	20×20	1.8×	N/A
	30×30	1.9×	N/A
#4	10×10	1.6×	2.2×
	20×20	1.9×	2.7×
	30×30	1.9×	3.3×

Table 5.5 Average speed-up achieved by the best configuration (i.e., platform #4 exploiting 4 GPUs) against the other configurations.

Platform	**Model size**	**Average speed-up**	
		1 GPU	**2 GPUs**
#1	10×10	1.9×	N/A
	20×20	2.1×	N/A
	30×30	2.6×	N/A
#2	10×10	1.8×	N/A
	20×20	1.9×	N/A
	30×30	2.4×	N/A
#3	10×10	1.6×	1.0×
	20×20	2.0×	1.1×
	30×30	3.2×	1.7×
#4	10×10	2.2×	1.3×
	20×20	2.7×	1.4×
	30×30	3.3×	1.8×

both in the case of single- and multi-GPU configurations, a very low statistical dispersion especially in the 10×10 and 20×20 models, with values considerably concentrated around the median. On the other hand, in the case of the model with 30 chemical reactions and molecular species, the performance of MS^{2}PSO executed on platform #2 are in general better than the other single-GPU configurations.

When considering configurations in which 2 or 4 GPUs are exploited in parallel, we observe that the average running times are strongly reduced, confirming the efficiency of the Master-Slave implementation of MS^{2}PSO. These results are also confirmed by the values reported in Table 5.4, where we calculated the speed-up achieved by multi-GPU configurations of platforms #3 and #4 with respect to the single-GPU configurations of the same platforms. For instance, by exploiting 4 GTX Titan X GPUs, the running time of the PE can be reduced more than three times, with respect to the same platform exploiting a single GPU. Finally, in Table 5.5 we calculated the speed-up achieved by the 4-GPUs configuration of platform #4—which obtained the lowest average running times for the PE of all models—with respect to all other possible configurations. In this case, MS^{2}PSO allows for achieving more than $3\times$ speed-up with respect to the other platforms. We can then argue that MS^{2}PSO scales well with the size of the model, especially when 4 GPUs are exploited for the PE; indeed, the running times obtained with this configurations with different models are comparable, as shown by the boxplots of Figure 5.33.

5.3.3 Conclusions

We presented MS^{2}PSO, a parallel and distributed implementation of a multi-swarm version of PSO to efficiently tackle the PE of biochemical systems. Considering the typical scenario of biological laboratory research, in which multiple target data related to different experimental conditions are available, MS^{2}PSO is capable of estimating a set of kinetic parameters that correctly reproduces the dynamics of the systems in all conditions. MS^{2}PSO is based on the Master-Slave distributed computing paradigm, in which the Master process offloads the time-consuming calculations needed to carry out the PE tasks onto different Slaves processes; in addition, each Slave exploits cupSODA [306], which allows for running in a parallel fashion on the cores of the GPU the simulations necessary to calculate the fitness values required for the optimization.

We evaluated the performance of MS^{2}PSO by executing the PE of three synthetic models characterized by 10, 20, 30 parameters to be estimated, using four computational platforms equipped with different Nvidia GeForce GPUs. The results highlighted that platforms in which 2 or 4 GPUs are exploited in parallel, the average running times for the PE are strongly reduced, revealing the effectiveness of the Master-Slave implementation. In particular, MS^{2}PSO achieved more than $3\times$ speed-up with respect to the single-GPU configurations. MS^{2}PSO will be integrated in the forthcoming web-based computational platform for Systems Biology COSYS [95], in order to provide an efficient means for the estimation of the kinetic constants in models of biological systems.

5.4 PE of a human intracellular metabolic pathway

Considering the results discussed in Section 5.2.4, where we showed that FST-PSO is capable of outperforming other metaheuristics in solving the PE problem of small and medium scale models, we applied it here to perform the PE of the model of the human intracellular metabolic pathway in a red blood cell described in Appendix A.3, in which 78 parameters need to be estimated.

Metabolic networks are inherently complex systems, characterized by thousands of reactions and metabolites, and regulated by a number of other intra- and extra-cellular processes [37, 287]. Kinetic models of metabolism exploit mechanistic biological information to provide a detailed description of the reaction network, as well as to simulate or predict its dynamic behavior in different conditions. To this aim, both the structure of the network—that is, how metabolites interact with enzymes, operating either as reactants, products, inhibitors, or activators in each reaction—and the kinetic parameters need to be properly defined. Unfortunately, due to the high costs and complexity of experimental procedures, these data are scarce or incomplete, and the resulting model is usually undetermined [71]. In addition, the modeling task is further complicated by the fact that the flux of metabolites through the network strongly depends on many interrelated processes, like transcription, translation, post-translational modifications, and allosteric control [287]. As a matter of fact, many sources of indetermination can affect models of uncharacterized enzyme isoform mixtures [401]. Enzyme isoforms, also called isozymes, are structurally similar but non-identical protein complexes that are able to catalyze the same biochemical reaction. Every cell type presents specific mixtures of isoforms of its metabolic enzymes and, in order to quantify their abundance, experimental techniques, like proteomics analyses, are used [446]. The structural differences of these proteins result in different kinetic behaviors of the catalytic process [328, 472]. However, the main databases that collect information about enzyme kinetics and the associated parameters, such as SABIO-RK [469] and BRENDA [387], generally lack details when it comes to differences among isozymes. The knowledge so far available in the literature thus prevents modelers from formulating a precise mathematical representation of the kinetic differences of isozymes. As a result, the vast majority of published metabolic models do not explicitly take these kinetic differences into account [401]. Instead, the averaged behavior of the whole isoform mixture is reproduced using a single set of kinetic parameters, either taken from the literature or inferred through global optimization methods. The main drawback of this assumption emerges, in particular, when the two following conditions superimpose: (*i*) the isozymes of the modeled metabolic system display significant kinetic variations; (*ii*) in the modeled experimental condition the abundance of isozymes differs from the condition in which the kinetic parameters of the whole mixtures were originally estimated. These issues should be carefully taken into consideration since the expression levels of metabolic enzymes change due to genetic knock-downs, therapeutic interventions, as well as various environmental stimuli [287]; therefore,

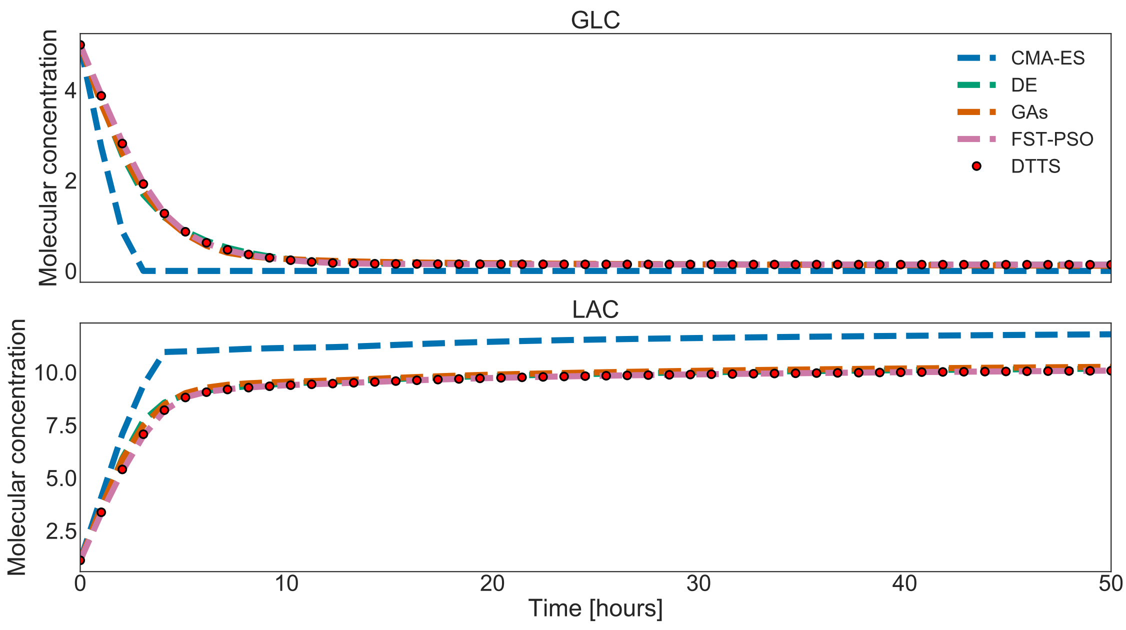


Fig. 5.34 Dynamics of the GLC (*top*) and LAC (*bottom*) species of the human intracellular metabolic network, obtained with the best parameterization found by CMA-ES, DE, GAs and FST-PSO, compared with the experimental DTTS.

the modeling and understanding of metabolic systems in perturbed conditions can highly benefit from the explicit representation of enzyme isoforms.

Here, we apply FST-PSO to calibrate the model of the red blood cell metabolism presented in [216] (see Appendix A.3), considering the different isoforms of the enzyme hexokinase (HK), the first enzyme of the glycolythic pathway that converts glucose (GLC) into glucose-6-phosphate (G6P). We decided to explicitly represent the three isoforms of HK that are the most abundant across different cell types and are known to have different kinetic properties [467]. The resulting RBM is composed of 114 species and 226 reactions; therefore, to drastically reduce the running time required to execute the simulations of this large-scale model, we exploit FiCoS (see Section 4.3).

5.4.1 Results and discussion

In this section we present the results of the PE of the model of the red blood cell metabolism; to show the effectiveness of FST-PSO we compared its performance against CMA-ES, DE and GAs. In all tests, the initial population of each metaheuristic was randomly generated by sampling the parameters of each individual by using a logarithmic distribution [73], restricting the values in the boundaries of the search space. The simulations required for the fitness calculation were performed using absolute error tolerance 10^{-8} and relative error tolerance 10^{-4}. All tests that follow were repeated 20 times, to collect statistical information and assess the ABF value, in order to discuss the average behavior of each algorithm.

The model considered here is an extension of the model proposed by Jamshidi and Palsson [216], in which we increased the level of detail by specifying the three main isoforms of HK enzyme (i.e., HK I, II and III). For our experiments we assumed the relative abundances of HK I, II and III equal to 0.6, 0.3 and 0.1, respectively We point out that these values were not fixed according to some experimental measurements, and thus are meant to represent a *possible* condition for the metabolic network. By so doing, we included 78 new reactions, whose kinetic constants are unknown.

To perform the PE we relied on synthetic experimental measurements of GLC and lactate (LAC) concentrations in a 50 hours time window. The ranges of feasible parameter values were defined by considering the 26 original parameter values used to generate the synthetic data, and allowing for a variation of at most 3 orders of magnitude above and below these values. The complete RBM—including the known and estimated parameters, and the initial conditions—is shown in Appendix A.3.

Figure 5.34 shows the comparison between the DTTS (red dots) and the simulation of GLC and LAC obtained with the best parameterizations found by CMA-ES (blue line), DE (green line), GAs (orange line), and FST-PSO (pink line). We observe that all metaheuristics, except for CMA-ES, achieved a perfect fitting at the end of the optimization process. Figure 5.35 reports the ABF calculated according to the results of 20 independent PE repetitions, showing that FST-PSO is capable of outperforming the other metaheuristics for this specific task, confirming the results presented in Section 5.2.4. In order to validate the findings shown in Section 5.2.4, we executed the PE of this model by applying the logarithmic transformation to CMA-ES and FST-PSO (no $\mathbf{v}_{min}$). The results shown in Figure 5.36 confirm that the standard CMA-ES version (blue solid line) is not capable of achieving good optimization results in terms of ABF. On the contrary, thanks to the transformation in Equation 5.3, CMA-ES (blue dashed line) is capable of achieving results similar to FST-PSO (no $\mathbf{v}_{min}$) (magenta lines), in accordance with the general patterns observed with the tested synthetic models (see Section 5.2.4). In addition, it is worth noting that CMA-ES-log, on average, begins the optimization with a better initial distribution with respect to FST-PSO (no $\mathbf{v}_{min}$): indeed, the ABF in the case of CMA-ES-log at iteration 0 is approximately 25, while in the case of FST-PSO (no $\mathbf{v}_{min}$) is approximately 32. This result highlights a further advantage of our alternative representation of parameters.

From the computational perspective, FiCoS strongly reduced the running time required by the PE task, achieving a 30$\times$ speed-up running on a Nvidia GeForce Titan X (3072 cores,

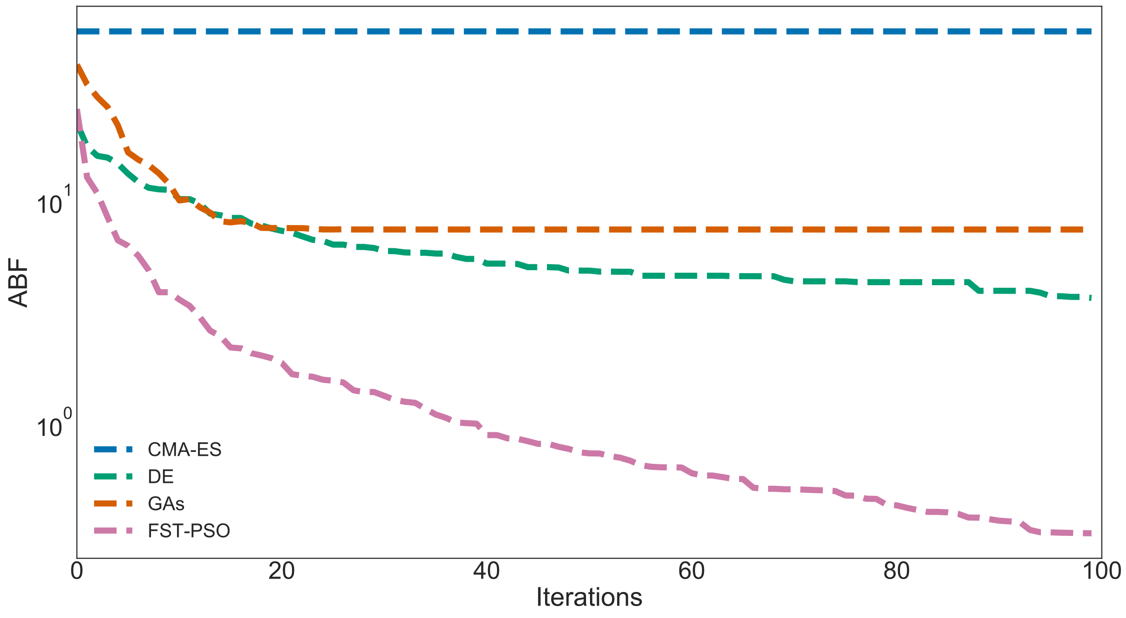


Fig. 5.35 ABF calculated running CMA-ES, DE, GA and FST-PSO for 20 independent repetitions. FST-PSO clearly outperforms the competitor methods.

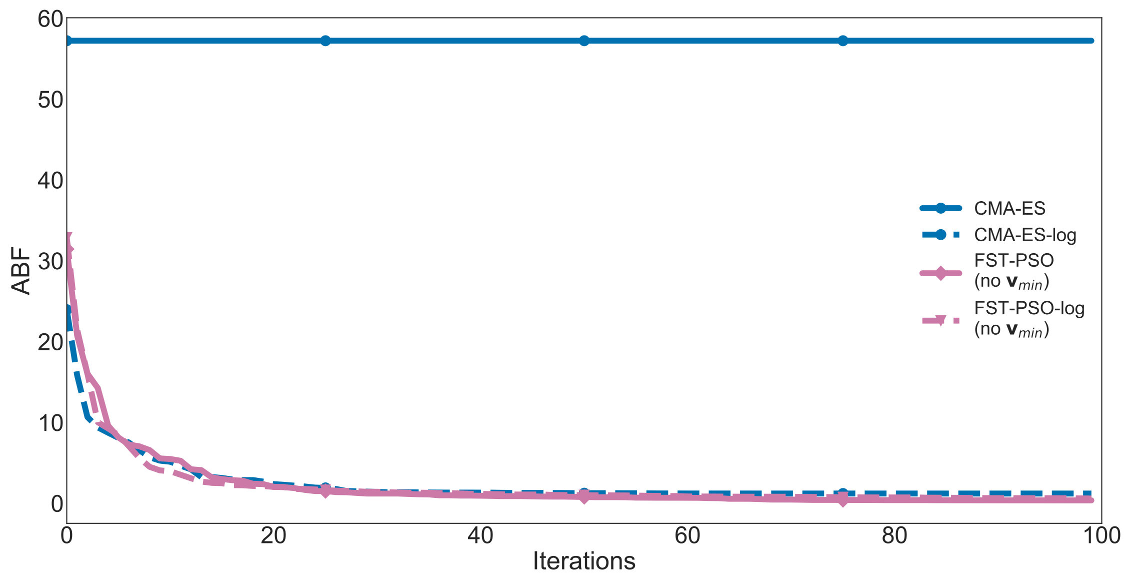


Fig. 5.36 ABF calculated running CMA-ES and FST-PSO (no $\mathbf{v}_{min}$) with normal and logarithmic semantics of parameters for 20 independent repetitions.

clock 1.075 GHz and RAM 12 GB) with respect to the same analysis carried out with the ODE solver LSODA (see Section 1.2.1), running on a CPU Intel Core i7-2600 (3.4 GHz, RAM 8 GB).

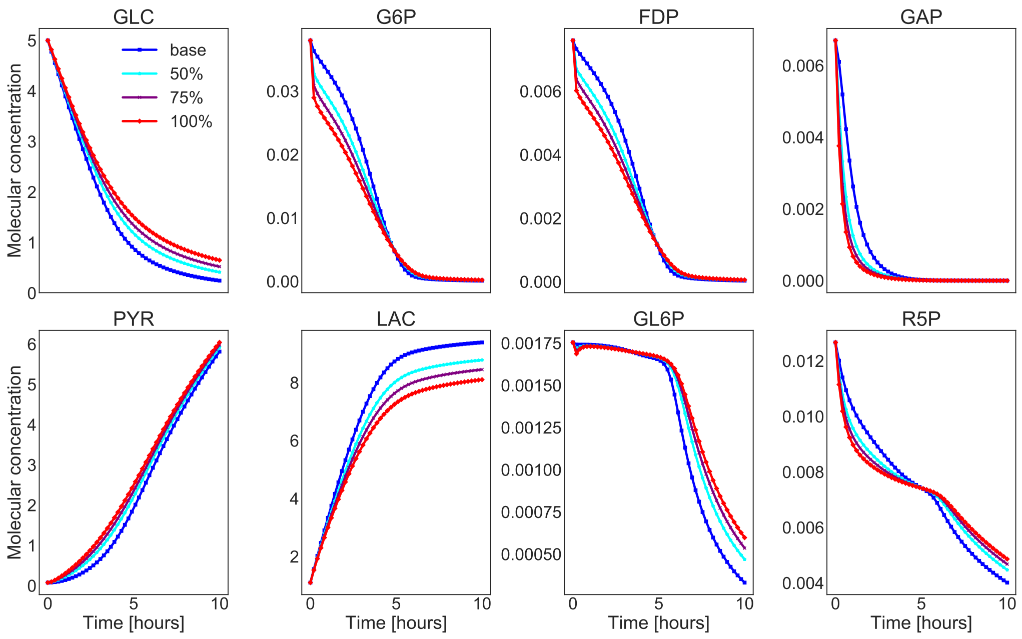


Fig. 5.37 Dynamic profiles of metabolite concentration simulated with the baseline model, and with increasing (50, 75, 100%) knock-down interventions on the HK I isoform, in a 10 hours time window. GLC and G6P are upstream metabolites with respect to HK. Fructose-diphosphate (FDP), glyceraldehyde-3-phosphate (GAP), pyruvate (PYR) and LAC are downstream metabolites with respect to HK. 6-Phosphogluconolactone (GL6P) and ribose 5-phosphate (R5P) are involved in the pentose phosphate pathway.

We then explored how the model could be used to reproduce the effect of an isoform-specific modification of the system. This scenario can represent a gene knock-down experiment, the effect of a drug with an isoform-specific target, or a change in isozyme expression after the cell is exposed to an environmental stimulus [287]. In Figure 5.37 we show that 50%, 75% and 100% reductions of the concentration of the HK isoform (see Table A.6) with the highest abundance (0.6) affect the dynamics of key metabolites in the network in a 10 hours time window. Noteworthy, from these results we can see that a complete knock-out of one isoform is not necessarily detrimental for intracellular energy-producing pathways, as other isoforms with efficient kinetics can provide alternative catalytic routes. The simulations of the model in a 50 hours time window indeed show that the 50%, 75% and 100% knock-down interventions only result in negligible effects on steady state metabolite concentrations (see Figure 5.38).

The dynamics of GLC and LAC apparently display the most evident differences in a 50 hours window; however, it should be considered that this result was indeed expected as these two metabolites represent, respectively, the "source" and the "sink" entities in the

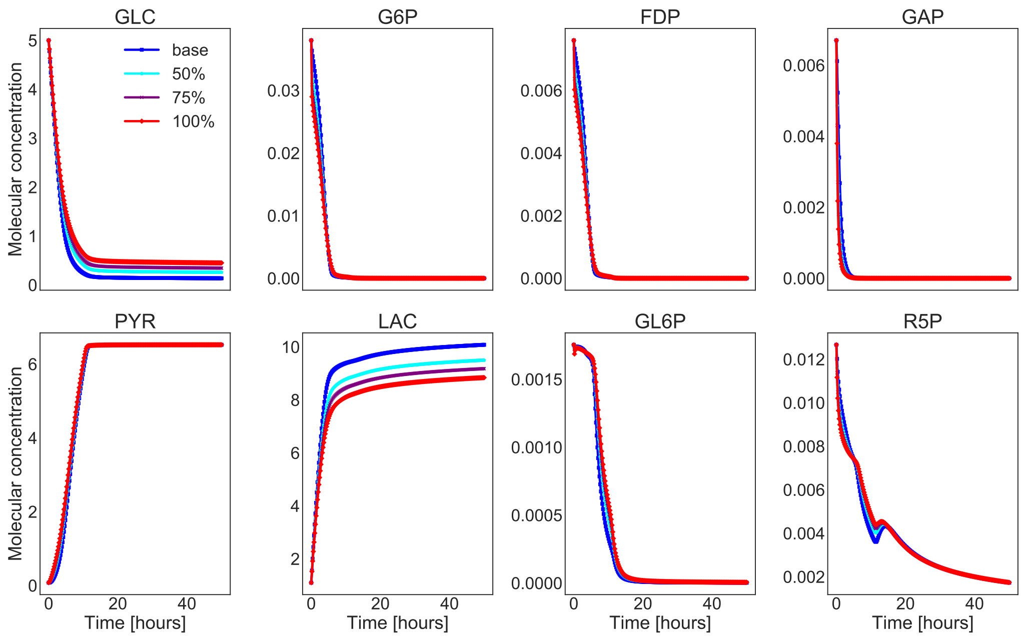


Fig. 5.38 Dynamic profiles of metabolite concentration simulated with the baseline model, and with increasing $(50, 75, 100\%)$ knock-down interventions on the HK I isoform, in a 50 hours time window. GLC and G6P are upstream metabolites with respect to HK. Fructose-diphosphate (FDP), glyceraldehyde-3-phosphate (GAP), pyruvate (PYR) and LAC are downstream metabolites with respect to HK. 6-Phosphogluconolactone (GL6P) and ribose 5-phosphate (R5P) are involved in the Pentose Phosphate Pathway.

biochemical network and no influx/efflux reactions were modeled. If we compare the GLC and LAC dynamics, it is anyway interesting to observe that altering the activity of enzymes like HK, which is known to exert high control over the whole pathway [401], produces proximal effects (upstream metabolite GLC is directly degraded by HK) and distal effects (9 metabolic reactions separate HK from the downstream metabolite LAC) on a comparable scale.

5.4.2 Conclusions

When mathematical models are used to investigate the effects produced by the different kinetic properties of metabolic isozymes, issues of lack of parameters and efficient simulation of the dynamics often arise. Here, we showed that FST-PSO coupled with FiCoS can effectively and efficiently solve the PE problem of large-scale models, thanks to the parallelization of the computations on the GPU. This is particularly well suited to deal with complex metabolic models that, for instance, include many reactions alternatively catalyzed by various isozymes with unknown kinetic parameters. As a matter of fact, we successfully estimated the 78 missing parameters related to HK isozymes. These allowed us to correctly reproduce the network behavior and to perform new *in silico* experiments. Thanks to FiCoS, we achieved a 30$\times$ speed-up with respect to the same methodology exploiting LSODA as ODE solver running on CPU.

Chapter 6

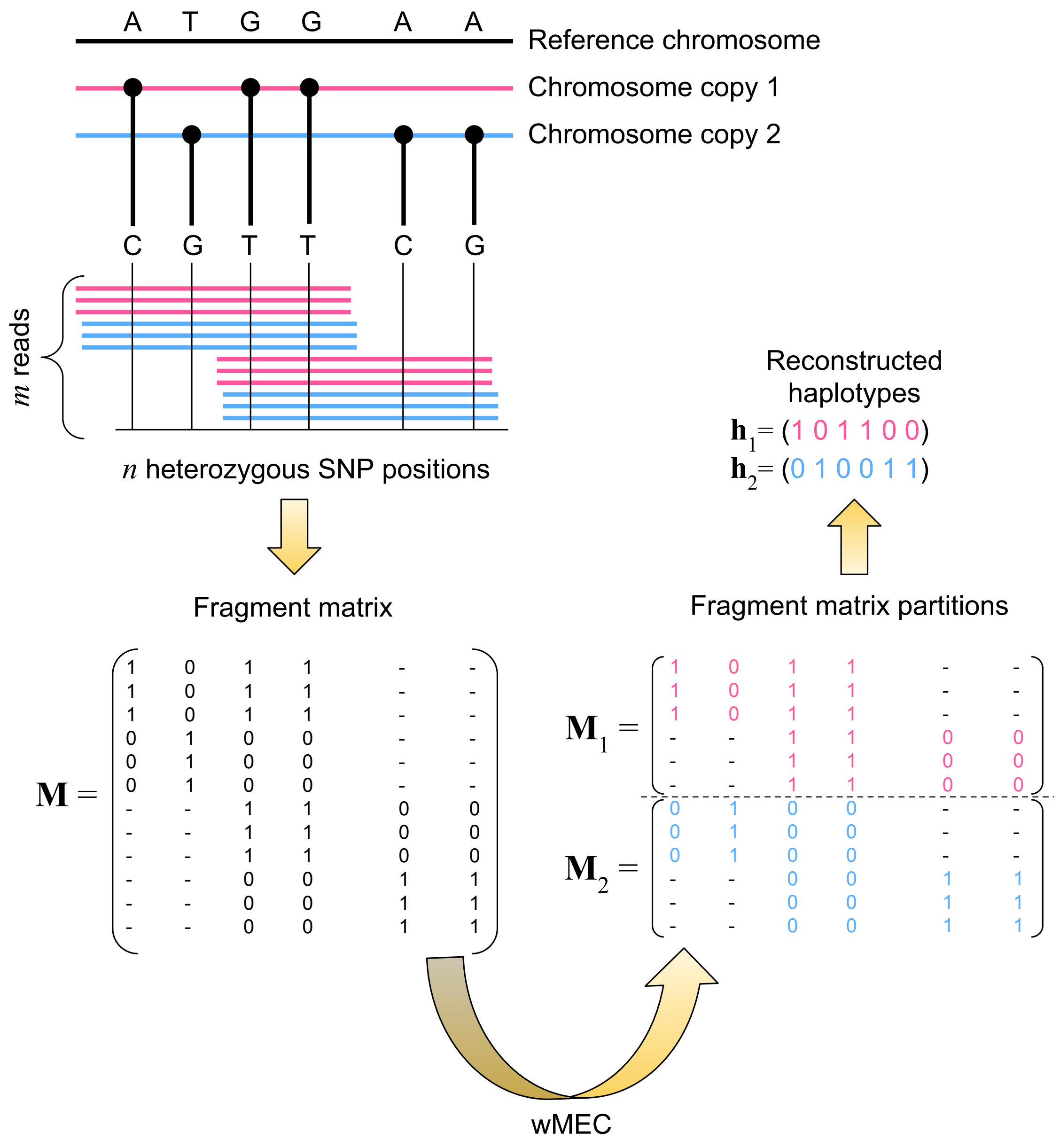
Computational method based on Genetic Algorithms for Haplotype Assembly

In order to deal with the computational hardness required by the Haplotype Assembly problem, we designed GenHap [433, 431], a computational method based on GAs that can efficiently solve large instances of the weighted Minimum Error Correction (wMEC) problem, yielding optimal solutions by means of a global search process, without any *a priori* hypothesis about the sequencing error distribution in the reads. The computational complexity of the problem is overcome by relying on a *divide-et-impera* approach, which provides faster and often more accurate solutions compared with the state-of-the-art haplotyping tools.

6.1 Problem formulation

Given n positions on two homologous sequences belonging to a diploid organism and m reads obtained after a sequencing experiment, we can reduce each read to a fragment vector $\mathbf{f} \in \{0, 1, -\}^n$, where 0 denotes a position that is equal to the reference sequence, 1 denotes a Single Nucleotide Polymorphism (SNP) with respect to the reference sequence and $-$ indicates a position that is not covered by the read. We define a haplotype as a vector $\mathbf{h} \in \{0, 1\}^n$, that is, the combination of SNPs and wild-type positions belonging to one of the two chromosomes. Given the two haplotypes $\mathbf{h}_1$ and $\mathbf{h}_2$—which refer to the first and second copy of the chromosome, respectively—a position j (with $j \in \{1, \dots, n\}$) is said to be heterozygous if and only if $h_{1_j} \neq h_{2_j}$, otherwise j is homozygous.

Let $\mathbf{M}$ be the "fragment matrix", that is, the $m \times n$ matrix containing all fragments. Two distinct fragments $\mathbf{f}$ and $\mathbf{g}$ are said to be in conflict if there is a position j (with $j \in \{1, \dots, n\}$) such that $f_j \neq g_j$ and $f_j, g_j \neq -$, otherwise they are in agreement. $\mathbf{M}$ is conflict-free if there



$$
\mathbf{M} = \begin{pmatrix}
1 & 0 & 1 & 1 & - & - \\
1 & 0 & 1 & 1 & - & - \\
1 & 0 & 1 & 1 & - & - \\
0 & 1 & 0 & 0 & - & - \\
0 & 1 & 0 & 0 & - & - \\
0 & 1 & 0 & 0 & - & - \\
- & - & 1 & 1 & 0 & 0 \\
- & - & 1 & 1 & 0 & 0 \\
- & - & 1 & 1 & 0 & 0 \\
- & - & 0 & 0 & 1 & 1 \\
- & - & 0 & 0 & 1 & 1 \\
- & - & 0 & 0 & 1 & 1
\end{pmatrix}
\quad
\mathbf{M}_1 = \begin{pmatrix}
1 & 0 & 1 & 1 & - & - \\
1 & 0 & 1 & 1 & - & - \\
1 & 0 & 1 & 1 & - & - \\
- & - & 1 & 1 & 0 & 0 \\
- & - & 1 & 1 & 0 & 0 \\
- & - & 1 & 1 & 0 & 0
\end{pmatrix}
\quad
\mathbf{M}_2 = \begin{pmatrix}
0 & 1 & 0 & 0 & - & - \\
0 & 1 & 0 & 0 & - & - \\
0 & 1 & 0 & 0 & - & - \\
- & - & 0 & 0 & 1 & 1 \\
- & - & 0 & 0 & 1 & 1 \\
- & - & 0 & 0 & 1 & 1
\end{pmatrix}
$$

Fig. 6.1 Simplified workflow of the Haplotype Assembly process. Raw sequencing data are initially aligned, defining m reads. Every position of the two chromosome copies is compared against a reference chromosome. The black solid points denote n heterozygous positions, along with the corresponding nucleobases. The fragment matrix $\mathbf{M}$ is defined assigning 1 to SNP positions and 0 to wild-type positions. To reconstruct the two haplotypes $\mathbf{h}_1$ and $\mathbf{h}_2$ characterized by the least number of corrections to the SNP values among the 2^n candidate haplotypes, the wMEC problem is solved by partitioning the matrix $\mathbf{M}$ into two disjoint matrices $\mathbf{M}_1$ and $\mathbf{M}_2$.

are two different haplotypes $\mathbf{h}_1$ and $\mathbf{h}_2$ such that each row M_i (with $i \in \{1,\ldots,m\}$) is in agreement with $\mathbf{h}_1$ or $\mathbf{h}_2$. The overall Haplotype Assembly process is outlined in Figure 6.1.

We can extend the heterozygous and homozygous definitions at the column level as follows: a column c of $\mathbf{M}$ is homozygous if all its values are either in $\{0,-\}$ or in $\{1,-\}$, on the contrary c is heterozygous because its values are in $\{0,1,-\}$, meaning that both a SNP and a wild-type exist in that position. Finally, we can detect the case where two distinct fragments are in conflict and measure their diversity by defining a distance $D(\cdot,\cdot)$ that calculates the number of different values between two fragments. Namely, given two fragment $\mathbf{f} = (M_{i1},\dots,M_{in})$ and $\mathbf{g} = (M_{l1},\dots,M_{ln})$ of $\mathbf{M}$ (with $i,l \in \{1,\dots,m\}$), we consider:

$$D(\mathbf{f},\mathbf{g}) = \sum_{j=1}^{n} d(f_j,g_j), \tag{6.1}$$

where $d(f_j,g_j)$ is defined as:

$$d(x,y) = \begin{cases} 1, & \text{if } x \neq y,\ x \neq -,\ \text{and } y \neq - \\ 0, & \text{otherwise} \end{cases}. \tag{6.2}$$

Equation 6.1 defines the *extended Hamming distance* between two ternary strings $\mathbf{f}$ and $\mathbf{g}$ [81], denoting the total number of positions wherein both characters of $\mathbf{f}$ and $\mathbf{g}$ belong to $\{0,1\}$, but they are different according to Equation 6.2.

If $\mathbf{M}$ is conflict-free, then it can be partitioned into two disjoint matrices $\mathbf{M}_1$ and $\mathbf{M}_2$, each one containing a set of conflict-free fragments. We can infer the two haplotypes $\mathbf{h}_1$ and $\mathbf{h}_2$ from $\mathbf{M}_1$ and $\mathbf{M}_2$, respectively, as follows:

$$h_{k_j} = \begin{cases} 1, & \text{if } N_{1_j}(\mathbf{M}_k) \geq N_{0_j}(\mathbf{M}_k) \\ 0, & \text{otherwise} \end{cases}, \tag{6.3}$$

where $j \in \{1,\dots,n\}$, $k \in \{1,2\}$, and $N_{0_j}(\mathbf{M}_k)$, $N_{1_j}(\mathbf{M}_k)$ denote the number of 0s and 1s in the j-th column, respectively. In such a way, $\mathbf{N}_0(\mathbf{M}_k)$ is the vector consisting of the number of 0s of each column j using the reads of the partition $\mathbf{M}_k$, while $\mathbf{N}_1(\mathbf{M}_k)$ is the vector consisting of the number of 1s of each column j represented by the partition $\mathbf{M}_k$.

In order to solve the wMEC problem, $\mathbf{N}_0$ and $\mathbf{N}_1$ are calculated by using the $m \times n$ weight matrix $\mathbf{W}$ representing the weight associated with each position in each fragment. As a matter of fact, $\mathbf{W}$ can be divided into the two disjoint partitions $\mathbf{W}_1$ and $\mathbf{W}_2$, whose row indices correspond to those in $\mathbf{M}_1$ and $\mathbf{M}_2$, respectively. We can extend Equation 6.3 taking into account the weights as follows:

$$h_{k_j} = \begin{cases} 1, & \text{if } N_{1_j}(\mathbf{W}_k) \geq N_{0_j}(\mathbf{W}_k) \\ 0, & \text{otherwise} \end{cases}, \tag{6.4}$$

where $j \in \{1,\dots,n\}$, $k \in \{1,2\}$, and $N_{0_j}(\mathbf{W}_k)$, $N_{1_j}(\mathbf{W}_k)$ denote the sum of the weights associated with the 0 and 1 elements in the j-th column, respectively.

The distance $D(\cdot,\cdot)$ given in Equation 6.1 can be used also to evaluate the distance between a fragment and a haplotype, by means of the following error function:

$$\mathcal{E}(\mathbf{M}_1,\mathbf{M}_2,\mathbf{h}_1,\mathbf{h}_2) = \sum_{k=1}^{2} \sum_{\mathbf{f}\in\mathbf{M}_k} D(\mathbf{f},\mathbf{h}_k). \tag{6.5}$$

The best partitioning of $\mathbf{M}$ can be obtained by minimizing Equation 6.5, inferring $\mathbf{h}_1$ and $\mathbf{h}_2$ with the least number of errors. Equation 6.5 is used as fitness function in GenHap.

6.2 Implementation strategy

GenHap exploits a very simple and efficient structure for the individuals composing the population P of the Genetic Algorithm (GA) at the basis of GenHap. This structure encodes a partition of the fragment matrix $\mathbf{M}$ as a binary string. In particular, each individual $C_p = [C_{p_1}, C_{p_2}, \dots, C_{p_m}]$ (with $p \in \{1,\dots,|P|\}$) is encoded as a circular array of size m (i.e., the number of reads). In order to obtain the two partitions $\mathbf{M}_1$ and $\mathbf{M}_2$, C_p is evaluated as follows: if the i-th bit is equal to 0, then the read i belongs to $\mathbf{M}_1$; otherwise, the read i belongs to $\mathbf{M}_2$. Once the two partitions are computed, GenHap infers the haplotypes $\mathbf{h}_1$ and $\mathbf{h}_2$ by applying Equation 6.4. Finally, Equation 6.5 is exploited to calculate the number of errors made by partitioning $\mathbf{M}$ as encoded by the individuals in P. This procedure is iterated until the maximum number of iterations T is reached, the number of errors is equal to 0 or the fitness value of the best individual does not improve for $\theta = \lceil 0.25 \cdot T \rceil$ iterations.

Among the different selection mechanisms employed by GAs (e.g., roulette wheel [163], ranking [26], tournament [293]), GenHap exploits the tournament selection to create an intermediate population P', starting from P. In each tournament, κ individuals are randomly selected from P and the individual characterized by the best fitness value is added to P'. The size of the tournament κ is related to the selection pressure: if κ is large, then the individuals characterized by worse fitness values have a low probability to be selected, therefore the variability of P' might decrease.

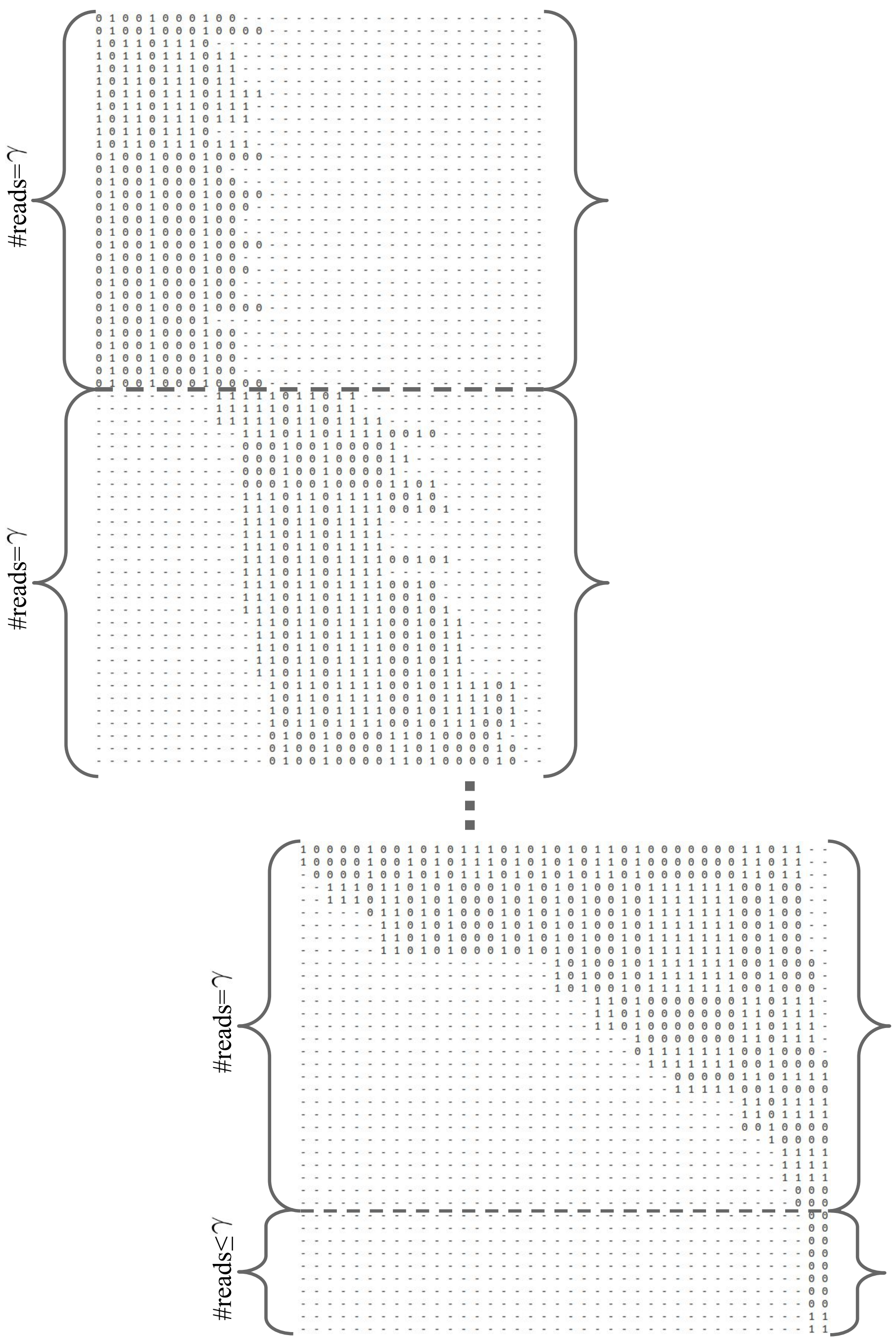


Fig. 6.2 Scheme of the partition of the input matrix: the input matrix $\mathbf{M} \in \{0,1,-\}^{m\times n}$ is split into sub-matrices consisting of γ reads, generating $\Pi = \lfloor m/\gamma \rfloor$ sub-problems that are solved independently by a GA instance. The last sub-matrix could have a number of reads lower than γ.

Afterwards, the genetic operators (i.e., crossover and mutation) are applied to the individuals belonging to P' to obtain the offspring for the next iteration. GenHap exploits a single-point crossover with mixing ratio equal to 0.5. It is applied with a given probability c_r and allows for the recombination of two parent individuals $C_y, C_z \in P'$ (for some $y, z \in \{1, \ldots, |P|\}$), generating two offspring that possibly have better characteristics with respect to their parents.

In order to increase the variability of the individuals, one or more elements of the offspring can be modified by applying the mutation operator. GenHap makes use of a classic mutation in which the elements C_{p_e} (with $e \in \{1, \ldots, m\}$) of the individual can be flipped (i.e., from 0 to 1 or viceversa) with probability m_r. Besides this mutation operator, GenHap implements an additional bit-flipping mutation in which a random number of consecutive elements of the individual is mutated according to the probability m_r. This operator is applied if the fitness value of the best individual does not improve for a given number of iterations (2 in our tests).

Finally, to prevent the quality of the best solution from decreasing during the optimization, GenHap exploits an elitism strategy, so that the best individual from the current population is copied into the next population without undergoing the genetic operators.

Unlike the work in [457], GenHap solves the wMEC problem instead of the unweighted MEC formulation, by means of Equation 6.4. Moreover, differently from the other heuristic strategies, such as ReFHap [121] and ProbHap [239], we did not assume the all-heterozygosity of the phased positions [81]. Under this assumption, every column corresponds to heterozygous sites, implying that $\mathbf{h}_1$ must be the complement of $\mathbf{h}_2$. In addition, since the required execution time as well as the problem difficulty increase with the number of reads and SNPs, to efficiently solve the wMEC problem we split the fragment matrix $\mathbf{M}$ into $\Pi = \lfloor m/\gamma \rfloor$ sub-matrices consisting of γ reads (see Figure 6.2). Following a *divide-et-impera* approach [274], the computational complexity can be tackled by partitioning the entire problem into smaller and manageable sub-problems, each one solved by a GA that converges to a solution characterized by two sub-haplotypes with the least number of corrections to the SNP values. The solutions to the sub-problems achieved by the Π GA instances are finally combined. This approach is feasible thanks to the long reads with higher coverage produced by the second- and third-generation sequencing technologies. As a matter of fact, highly overlapping reads allow us to partition the problem into easier sub-problems, avoiding the possibility of obtaining incorrect reconstructions during the merging phase.

The parameter γ, used for the calculation of Π, depends on the coverage value and on the nature of the sequencing technology; its value must be correctly set to avoid discrete haplotype blocks that do not exist in the input matrix $\mathbf{M}$. Generally, the intervals where several independent historical recombination events occurred separate discrete blocks, revealing greater haplotype diversity for the regions spanning the blocks [99].

GenHap firstly detects all the haplotype blocks inside the fragment matrix **M** and then, in each block, it automatically sets γ equal to the mean coverage of that block to partition the reads. Notice that GenHap solves each block sequentially and independently, obtaining a number of haplotype pairs equal to the number of detected blocks. By so doing, for each block GenHap proceeds by executing Π different GA optimizations, one for each sub-problem, calculating $2 \cdot \Pi$ sub-haplotypes. The length of the individuals is equal to γ, except for the last sub-problem that could have a number of reads smaller than γ (accordingly, the length of the individuals could be smaller than γ).

Since the problem is divided into Π sub-problems, two sub-problems referring to contiguous parts of the two chromosome copies might contain some overlapped positions that can be either homozygous or heterozygous. However, the reads covering an overlapped position might not be entirely included in the same sub-problem. For this reason, during the GA-based optimizations, all the phased positions are assumed to be heterozygous. If a position j is homozygous (i.e., all the reads covering this position have the same value, belonging to $\{0,-\}$ or $\{1,-\}$, in both the sub-partitions and in every read covering it), then only one of the two sub-haplotypes will have the correct value. This specific value is correctly assigned to the sub-haplotype covered by the highest number of reads by following a majority rule. As soon as the two sub-haplotypes are obtained, all the possible uncorrected heterozygous sites are removed and the correct homozygous values are assigned by checking the columns of the two sub-partitions. Finally, once all sub-problems in Π are solved, GenHap recombines the sub-haplotypes to obtain the two entire haplotypes $\mathbf{h}_1$ and $\mathbf{h}_2$ of the block under analysis.

GenHap is also able to find and mask the ambiguous positions by replacing the 0 or 1 value with a X symbol. We highlight that an ambiguous position is a position covered only by the reads belonging to one of the two haplotypes.

In order to efficiently solve the wMEC problem and tackle its computational complexity, GenHap detects the haplotype blocks inside the matrix **M** and then, for each block, it splits the portion of **M** into Π sub-matrices consisting of γ reads. By so doing, the convergence speed of the GA is increased thanks to the lower number of reads to partition in each sub-problem with respect to the total number of reads of the whole problem. As shown in Figure 6.3, the Π sub-matrices are processed in parallel by means of a *divide-et-impera* approach that exploits a Master-Slave distributed programming paradigm [430] to speed-up the overall execution of GenHap. This strategy allowed us to distribute the computation in presence of multiple cores. As a matter of fact, GenHap works by partitioning the initial set of reads into sub-sets and solving them by executing different GA instances. This strategy can be exploited in GenHap, as it solves the wMEC problem working on the rows of the fragment

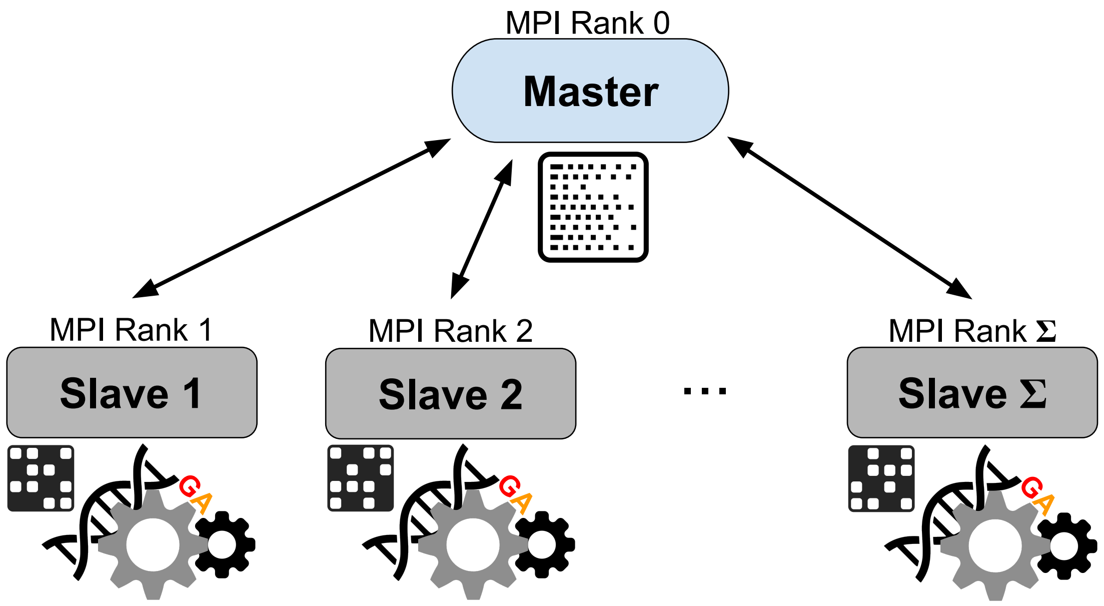


Fig. 6.3 Scheme of the Master-Slave implementation of GenHap: the Master process orchestrates all the Σ Slaves sending one or more sub-partitions to each Slave, which then solves the assigned wMEC sub-task.

matrix **M**; on the contrary, several state-of-the-art exact methods (e.g., HapCol [342]) work considering the columns of **M**, which cannot be independently processed in parallel.

The functioning of our Master-Slave implementation can be summarized as follows:

1. the Master allocates the resources and detects the haplotype blocks inside the fragment matrix. For each detected block, it partitions the portion of the matrix **M** into Π sub-matrices and offloads the data onto the available Σ Slaves (in real scenarios, $\Sigma \ll \Pi$). During this phase, each Slave generates the initial population of the GA;

2. the σ-th Slave (with $\sigma \in \{1,\dots,\Sigma\}$) executes the assigned wMEC sub-task, running the GA for either θ non-improving iterations or T maximum iterations, independently of the other Slaves;

3. the process is iterated until all the wMEC sub-tasks are terminated;

4. the Master recombines the sub-solutions received from the Slaves, and returns the complete wMEC solution for the block under analysis.

GenHap was entirely developed using the `C++` programming language, exploiting the Message Passing Interface (MPI) specifications to leverage multi-core Central Processing Units (CPUs).

6.3 Results

In this section we first describe the synthetic and real datasets used during the tests and present the results obtained to identify the best GA setting. Then, we discuss the performance achieved by GenHap with respect to HapCol [342], which was previously shown to be more efficient than the other existing methods for the Haplotype Assembly problem, both in terms of memory consumption and execution time. Finally, we show how GenHap performs on data produced by four different sequencing platforms, namely:

- Illumina NovaSeq (Illumina Inc., San Diego, CA, USA) [350]: the most used and widespread platform belonging to the class of second-generation sequencing technologies, able to produce a huge number of short and precise reads (up to 150bp);
- Roche/454 (Roche AG, Basel, Switzerland) [270]: a second-generation sequencing technology able to produce accurate and slightly longer reads than Illumina sequencers (up to 700bp);
- PacBio RS II (Pacific Biosciences of California Inc., Menlo Park, CA, USA) [358, 363]: a third-generation sequencing technology able to produce long reads (up to 30000bp);
- Oxford Nanopore Technologies (ONT) MinION (ONT Ltd., Oxford, United Kingdom) [212, 213, 391]: the latest developed third-generation sequencing technology, able to produce reads that are tens of kilobases long.

6.3.1 GenHap accuracy

The analyzed datasets

In order to test the performance of GenHap, we generated two synthetic (yet realistic) datasets, each one consisting of instances obtained from a specific sequencing technology. In particular, we considered the Roche/454 genome sequencer (Roche AG, Basel, Switzerland), representing one of the next-generation sequencing (NGS) systems able to produce long and precise reads, and the PacBio RS II sequencer [363, 66], which is an emerging third-generation sequencing technology. Note that the reads produced by the Roche/454 sequencer are approximately 9-times shorter than those generated by the PacBio RS II system.

In order to generate the datasets, we exploited the General Error-Model based SIMulator (GemSIM) toolbox [281]. GemSIM is a software able to generate *in silico* realistic sequencing data. It relies on empirical error models and distributions learned from real NGS data, and simulates both single- and paired-end reads from a single genome, collection of genomes, or set of related haplotypes. GemSIM can in principle simulate data from any sequencing technology producing output data encoded in the FASTQ format [89], for raw reads, and Sequence Alignment/Map (SAM), for aligned reads. We exploited the error model for the Roche/454 sequencer, already available in GemSIM, and defined an additional error model for the PacBio RS II technology. The synthetic reads were generated from the reference sequence of the human chromosome 22 (UCSC Genome Browser, GRCh37/hg19 Feb. 2009 assembly [68]), in which random SNPs were inserted.

We exploited the GemHaps tool included in GemSIM [281] to generate a haplotype file starting from a given genome sequence, and specifying the number as well as the frequency of SNPs in each haplotype, denoted by #SNPs and f_{SNPs}, respectively. Note that the SNP positions were randomly determined. Then, the resulting haplotype file was processed by GemReads, together with an error model file (generated by GemErr or supplied in GemSIM), a FASTA genome file (or directory), and the selected quality score offset. The resulting SAM file was converted into the compressed Binary Alignment/Map (BAM) format for a more efficient manipulation [258]. In order to store the SNPs, we exploited the Variant Call Format (VCF) [100], which is the most used format that combines DNA polymorphism data, insertions and deletions, as well as structural variants. Lastly, the BAM and VCF files were processed to produce a WhatsHap Input Format (WIF) file [333], which is the input of GenHap.

The two synthetic datasets are characterized by the following features: *i*) #SNPs $\in \{500, 1000, 5000, 10000, 20000\}$ (equally distributed over the two haplotypes); *ii*) coverage cov $\in \{\sim 30\times, \sim 60\times\}$; *iii*) average $f_{\text{SNPs}} \in \{100, 200\}$, which means one SNP every 100bp or 200bp [299, 140], varying the portion of genome onto which the reads were generated. Read lengths were set to 600bp and 5000bp for the Roche/454 and the PacBio RS II sequencers, respectively. The number of reads was automatically calculated according to the value of cov and the sequencing technologies, by means of the following relationship:

$$\#\text{reads} = \text{cov} \cdot \frac{len(\text{genome})}{len(\text{read})}, \tag{6.6}$$

where $len(\text{genome})$ represents the length of the considered genome, which starts at a given position x and ends at position $y = x + f_{\text{SNPs}} \cdot \#\text{SNPs}$.

Afterwards, to assess the performance of GenHap on real sequencing data, we exploited a WIF input file generated starting from high-quality SNP calls and sequencing data made publicly available by the Genome in a Bottle (GIAB) Consortium [493]. In particular, this data is produced with the PacBio technology and is limited to the chromosome 22 of the individual NA12878 (the real dataset is available in [38]) Moreover, we tested GenHap on

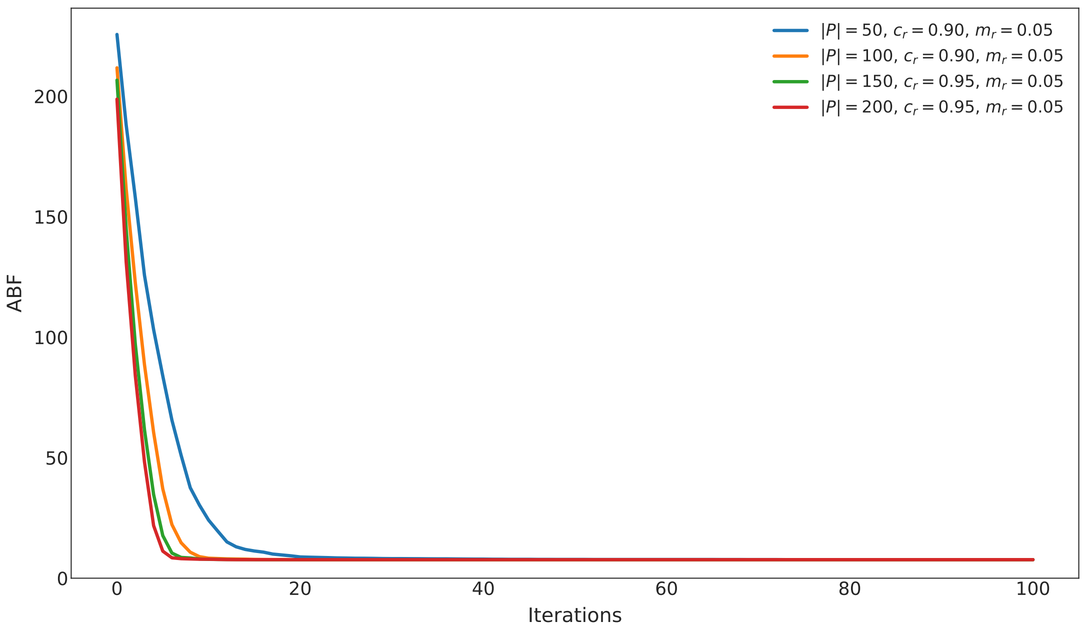


Fig. 6.4 Comparison of the ABF achieved by GenHap with the best parameterizations found for each value of $|P|$ tested here. The ABF was computed over the results of the optimization of instances characterized by #SNPs $\in \{500, 1000, 5000\}$ and $f_{\text{SNPs}} = 100$.

an additional [329], limiting our analysis to chromosome 22 as in the previous case. The available BAM file–containing long reads with high-coverage produced with the PacBio RS II sequencing technology–and the VCF file were processed to obtain a WIF input file as described above.

GA setting analysis

As a first step, the performance of GenHap was evaluated to determine the best settings for the Haplotype Assembly problem. We considered different instances for the two sequencing technologies employed (i.e., Roche/454 and PacBio RS II), and we varied the settings of GenHap used throughout the optimization process, as follows:

- size of the population $|P| \in \{50, 100, 150, 200\}$;
- crossover rate $c_r \in \{0.8, 0.85, 0.9, 0.95\}$;
- mutation rate $m_r \in \{0.01, 0.05, 0.1, 0.15\}$.

In all tests, the size of the tournament is fixed to $\kappa = 0.1 \cdot |P|$ and the maximum number of iterations is $T = 100$. A total of 6 different instances (3 resembling the Roche/454 sequencer and 3 the PacBio RS II sequencer) were generated by considering #SNPs $\in \{500, 1000, 5000\}$ and $f_{\text{SNPs}} = 100$.

We varied one setting at a time, leading to 64 different settings tested and a total number of $64 \times 6 = 384$ GenHap executions. These tests highlighted that, for each value of $|P|$, the best settings are:

1. $|P| = 50$, $p_c = 0.9$, $p_m = 0.05$;
2. $|P| = 100$, $p_c = 0.9$, $p_m = 0.05$;
3. $|P| = 150$, $p_c = 0.95$, $p_m = 0.05$;
4. $|P| = 200$, $p_c = 0.95$, $p_m = 0.05$.

Figure 6.4 shows the comparison of the performance achieved by GenHap with the settings listed above, where the Average Best Fitness (ABF) was computed by taking into account, at each iteration, the fitness value of the best individuals over the 6 optimization processes. Even though all settings allowed GenHap to achieve almost the same final ABF value, we observe that the convergence speed increases with the size of the population. However, also the running time of GenHap increases with the size of the population. In particular, the executions lasted on average 1.41 s, 2.33 s, 3.52 s, 4.95 s with $|P| \in \{50, 100, 150, 200\}$, respectively, running on one node of the Advanced Computing Center for Research and Education (ACCRE) at Vanderbilt University, Nashville, TN, USA. The node is equipped with 2 Intel® Xeon® E5-2630 v3 (8 cores at 2.40 GHz) CPUs, 240 GB of RAM and CentOS 7.0 operating system. To perform the tests we exploited all 8 physical cores of a single CPU.

Considering these preliminary results, we selected the parameter settings $|P| = 100$, $c_r = 0.9$, $m_r = 0.05$, as the best trade-off between convergence speed (in terms of ABF) and running time.

Comparison of the performance of GenHap and HapCol

The performance achieved by GenHap was compared with those obtained by HapCol [342], which was shown to outperform the main available haplotyping approaches. In particular, we exploited here a more recent version of HapCol capable of dealing with haplotype blocks [38]. The same computational platform used for the setting analysis of GenHap was used to execute all the tests on the two synthetic datasets described above.

We stress the fact that GenHap was compared against HapCol only on the instances with cov $\simeq 30\times$, since HapCol is not capable of solving instances with higher coverage values (i.e., the algorithm execution halts when a column covered by more than 30 reads is found).

Table 6.1 Comparison of GenHap and HapCol on Roche/454 dataset with cov $\simeq 30\times$. The performances were evaluated both in terms of *HE* and running time. The N/A symbol denotes that HapCol was not able to complete the execution on all the 15 instances.

			GenHap			HapCol		
f_{SNPs}	cov	#SNPs	Avg *HE*	Std dev *HE*	Avg Running Time [s]	Avg *HE*	Std dev *HE*	Avg Running Time [s]
100	$\sim 30\times$	500	0.04	0.08	0.21	0.00	0.00	0.62
		1000	0.09	0.08	0.36	0.00	0.00	1.20
		5000	0.18	0.06	3.17	0.01	0.03	5.35
		10000	2.50	5.52	10.33	6.55	16.38	10.23
200	$\sim 30\times$	500	0.09	0.14	0.34	0.00	0.00	0.50
		1000	0.09	0.10	0.63	0.01	0.03	0.96
		5000	3.61	3.43	6.07	0.38	0.78	4.90
		10000	2.15	1.62	17.24	N/A	N/A	N/A

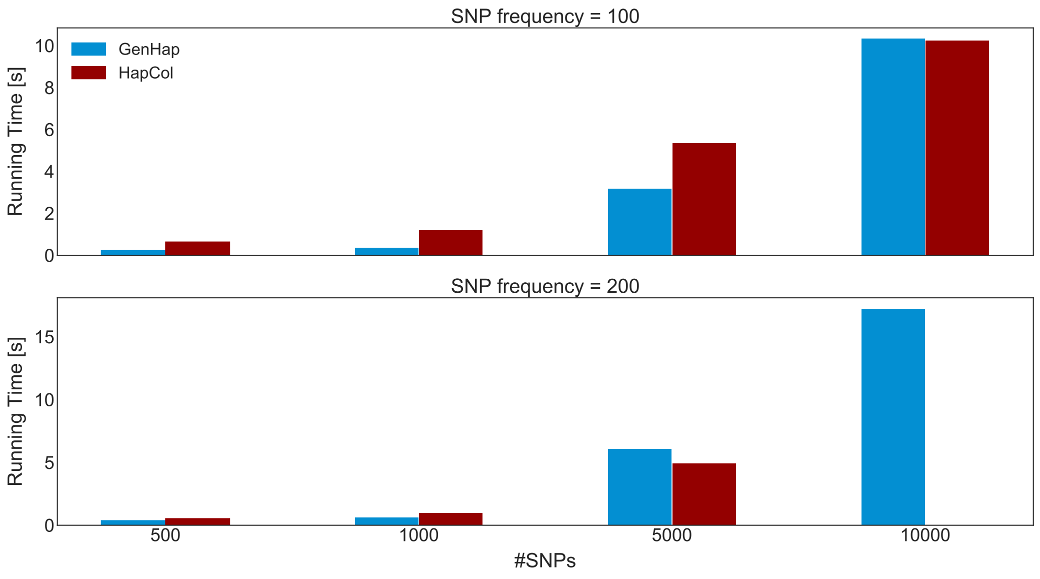


Fig. 6.5 Comparison of the average running time required by GenHap (blue bars) and HapCol (red bars) computed over 15 instances for each value of #SNPs $\in \{500, 1000, 5000\}$ obtained with the Roche/454 sequencing technology, cov $\simeq 30\times$ and $f_{\text{SNPs}} = 100$ (top) and $f_{\text{SNPs}} = 200$ (bottom). In the case of $f_{\text{SNPs}} = 200$ and #SNPs $= 10000$, HapCol was not able to complete the execution on all the 15 instances.

Considering the two sequencing technologies, we generated 15 different instances for each value of #SNPs and f_{SNPs}. The performance was then evaluated by computing (*i*) the average haplotype error rate (*HE*), which represents the percentage of SNPs erroneously assigned with respect to the ground truth [17], and (*ii*) the average running time.

As shown in Table 6.1, in the instances generated using the Roche/454 sequencing technology with $f_{\text{SNPs}} = 100$, both GenHap and HapCol reconstructed the two haplotypes, achieving an average *HE* lower than 0.2% with a negligible standard deviation in the case of #SNPs $\in \{500, 1000, 5000\}$. GenHap inferred the haplotypes characterized by 10000 SNPs with an average *HE* lower than 2.5% and a standard deviation around 5%, while HapCol obtained an average *HE* equal to 6.55% with a standard deviation around 16%. For what concerns the running time, GenHap outperformed HapCol in all tests except in the case of #SNPs = 10000, as shown in Figure 6.5, being around 4$\times$ faster in reconstructing the haplotypes. In the case of #SNPs = 10000, the running times are comparable, but GenHap obtains a lower *HE* than HapCol. In the instances generated using $f_{\text{SNPs}} = 200$ and #SNPs $\in \{500, 1000\}$, both GenHap and HapCol reconstructed the two haplotypes, achieving an average *HE* lower than 0.1% with a negligible standard deviation. When #SNPs $\in \{5000, 10000\}$ are taken into account, GenHap inferred the haplotype pairs with an average *HE* lower than 3.65% and a standard deviation lower than 3.5%. Notice that HapCol was not able to complete the execution on all the 15 instances characterized by 10000 SNPs. As in the case of instances with $f_{\text{SNPs}} = 100$, GenHap is faster than HapCol in all tests, except in the case of #SNPs = 5000. For what concerns the PacBio RS II sequencing dataset, since this technology is characterized by a higher error rate with respect to the Roche/454 sequencer, both GenHap and HapCol reconstructed the two haplotypes with higher *HE* values (see Table 6.2). Nonetheless, the average *HE* value is lower than 2.5% with a standard deviation lower than 1% in all cases. Figure 6.6 shows the running time required by GenHap and HapCol to reconstruct the haplotypes. As in the case of the Roche/r454 dataset, the running time increases with #SNPs, but GenHap always outperforms HapCol, achieving up to 20$\times$ speed-up.

Table 6.3 lists the results obtained by GenHap on the instances of the Roche/454 dataset characterized by cov $\simeq 60\times$, #SNPs $\in \{500, 1000, 5000, 10000\}$ and $f_{\text{SNPs}} \in \{100, 200\}$. In all tests with $f_{\text{SNPs}} = 100$, GenHap was always able to infer the two haplotypes with high accuracy, indeed the average *HE* values are always lower than 0.15%. In the instances generated with $f_{\text{SNPs}} = 200$, GenHap reconstructed the haplotype pairs with an average *HE* lower than 0.2%. This interesting result shows that higher coverages can help during the reconstruction phase, allowing GenHap to infer more precise haplotypes.

Regarding the PacBio RS II dataset, the achieved *HE* is on average lower than 1.25% with a standard deviation $\leq 0.4\%$ (see Table 6.4). In particular, the average *HE* decreases when the value of #SNPs or the coverage increase, thus suggesting that higher coverages values can considerably help in achieving a correct reconstruction of the two haplotypes. On the contrary, the running time increases at most linearly with respect to the coverage (see Table 6.4).

As a first test on real sequencing data, we exploited a WIF input file codifying the SNPs of the chromosome 22 generated from high-quality sequencing data made publicly available

Table 6.2 Comparison of GenHap and HapCol on PacBio RS II dataset with cov $\simeq 30\times$. The performances were evaluated both in terms of *HE* and running time.

			GenHap			HapCol		
f_{SNPs}	cov	#SNPs	Avg *HE*	Std dev *HE*	Avg Running Time [s]	Avg *HE*	Std dev *HE*	Avg Running Time [s]
100	$\sim 30\times$	500	2.04	0.59	0.11	2.42	0.78	2.24
		1000	1.27	0.51	0.19	1.20	0.61	1.89
		5000	1.06	0.19	0.94	0.60	0.17	9.04
		10000	0.96	0.19	2.50	0.43	0.11	15.51
		20000	1.02	0.14	8.49	0.41	0.11	31.13
200	$\sim 30\times$	500	2.09	0.52	0.14	1.73	0.42	0.95
		1000	1.70	0.24	0.22	1.09	0.41	1.84
		5000	1.05	0.18	1.39	0.54	0.11	7.10
		10000	1.13	0.18	4.09	0.51	0.17	14.13
		20000	1.02	0.13	13.86	0.33	0.05	27.55

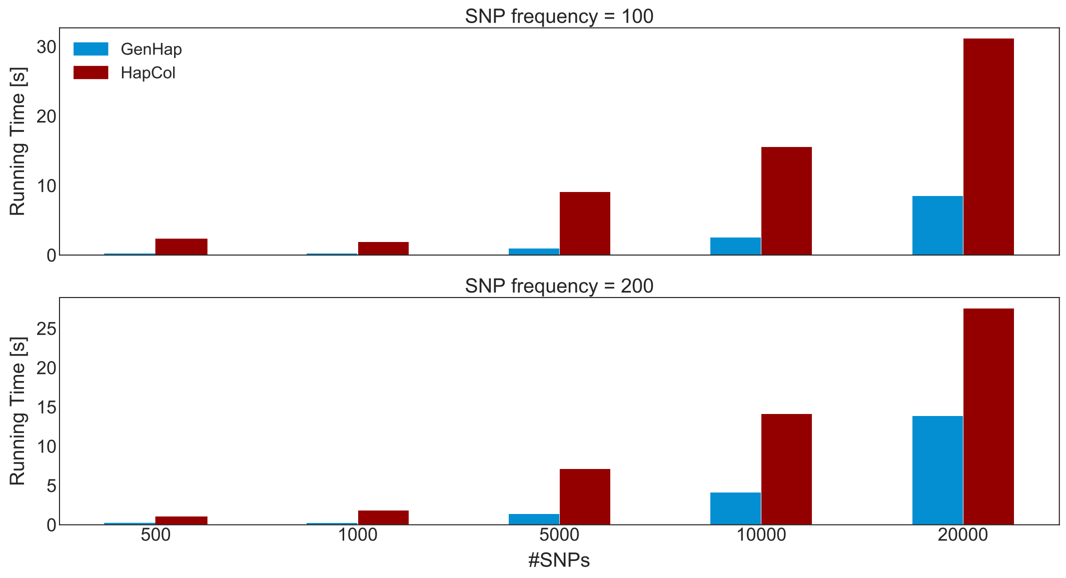


Fig. 6.6 Comparison of the average running time required by GenHap (blue bars) and HapCol (red bars) computed over 15 instances for each #SNPs $\in \{500, 1000, 5000, 10000, 20000\}$ obtained with the PacBio RS II sequencing technology, cov $\simeq 30\times$, $f_{\text{SNPs}} = 100$ (top) and $f_{\text{SNPs}} = 200$ (bottom).

by the GIAB Consortium. This instance contains #SNPs $\simeq 27000$ and #reads $\simeq 80000$ with average and maximum coverages equal to 22 and 25, respectively. In [38], in order to down-sample the instances to the target maximum coverages of $30\times$ allowed by HapCol, the authors applied a greedy-based pruning strategy. This procedure selects the reads characterized by high base-calling quality. GenHap detected and inferred the 305 different haplotype blocks in less than 10 minutes, obtaining approximately an 87% agreement with respect to the HapCol

Table 6.3 Results obtained by GenHap on Roche/454 dataset with cov $\simeq 60\times$. The performances were evaluated both in terms of *HE* and running time.

			GenHap		
f_{SNPs}	cov	#SNPs	Avg *HE*	Std dev *HE*	Avg Running Time [s]
100	$\sim 60\times$	500	0.00	0.00	0.26
		1000	0.05	0.05	0.54
		5000	0.10	0.03	6.57
		10000	0.15	0.03	21.13
200	$\sim 60\times$	500	0.00	0.00	0.37
		1000	0.07	0.09	0.89
		5000	1.13	1.72	11.17
		10000	2.00	1.02	53.77

Table 6.4 Results obtained by GenHap on PacBio RS II dataset with cov $\simeq 60\times$. The performances were evaluated both in terms of *HE* and running time.

			GenHap		
f_{SNPs}	cov	#SNPs	Avg *HE*	Std dev *HE*	Avg Running Time [s]
100	$\sim 60\times$	500	1.22	0.36	0.17
		1000	0.88	0.21	0.33
		5000	0.56	0.10	1.81
		10000	0.62	0.10	5.34
		20000	0.60	0.07	17.14
200	$\sim 60\times$	500	1.22	0.37	0.22
		1000	0.79	0.27	0.36
		5000	0.53	0.09	3.26
		10000	0.45	0.08	8.01
		20000	0.49	0.05	27.15

solution. This agreement was calculated considering every SNP of both haplotypes in each block.

We tested GenHap also on the chromosome 22 sequenced using the PacBio RS II technology (publicly available at [329]). This instance contains #SNPs $\simeq 28000$ and #reads $\simeq 140000$ with average and maximum coverages equal to 29 and 565, respectively. GenHap reconstructed the two haplotypes in about 10 minutes, showing its capability of dealing with instances characterized by high coverages, avoiding pruning pre-processing steps.

6.3.2 Computational performance

In order to assess the computational performance of GenHap, we considered different sequencing technologies, namely: Illumina NovaSeq, Roche/454, PacBio RS II, and ONT MinION. For each sequencing technology, we generated a single instance varying the following parameters:

- #SNPs $\in \{500, 1000, 5000, 10000, 20000\}$;

- cov $\in \{\sim 30\times, \sim 40\times, \sim 50\times, \sim 60\times\}$;
- average $f_{\text{SNPs}} = 200$ (i.e., one SNP every 200bp exists [299, 140]).

These instances were used to evaluate the scalability of GenHap by varying the number of cores, that is, #cores $\in \{2,4,8,16,24,32,40,48,56,64\}$. All tests were performed on the MARCONI supercomputer, which is based on Lenovo NeXtScale System® platform (Morrisville, NC, USA), provided by the Italian inter-university consortium CINECA (Bologna, Italy). Three different partitions running on CentOS 7.2 are available on this supercomputer:

$A1$ Broadwell (BDW) partition consists of 720 compute nodes, each one equipped with 2 Intel® Xeon® E5-2697 v4 (18 cores at 2.30GHz) and 128 GB RAM;

$A2$ Knights Landing (KNL) partition consists of 3600 compute nodes, each one equipped with an Intel® Knights Landing (68 cores at 1.40GHz and 16 GB MCDRAM), which is the next-generation of the Intel® Xeon Phi™ product family of many-core architecture, and 93 GB RAM;

$A3$ Skylake (SKL) partition consists of 92 compute nodes, each one equipped with 2 Intel® Xeon® 8160 CPU (18 cores at 2.10GHz) and 192 GB RAM.

Our analysis was carried out by using the computing nodes of partition $A2$, which was chosen due to the availability of a higher number of computing cores.

Figure 6.7 depicts the running times required by GenHap to infer the pairs of haplotypes. As expected, the processing time generally decreases along with the read length: according to Equation 1.22, the same coverage can be obtained by means of long reads coupled with a lower number of reads. This circumstance leads to a lower number of sub-problems to be solved, reducing the necessary computational effort. Moreover, the lowest running time is achieved on the instances generated relying on the ONT MinION, which is capable of producing long reads (up to 6000bp) with accuracy greater than 92%. As a matter of fact, the amount of SNPs to be corrected decreases when reads characterized by high accuracy are taken into account, allowing the GA instances to have a fast convergence to the optimal solutions. The results obtained for each sequencing platform are summarized as follows:

- Illumina NovaSeq: independently from the coverage, the lowest running time is achieved by exploiting 24 cores to parallelize the GA instances when #SNPs = 10000. When #SNPs < 10000, 16 or 24 cores require the minimum running time to infer the haplotype pairs;
- Roche/454: when #SNPs $\geq$ 5000, the best performance of GenHap is achieved by exploiting 16 or 24 cores, otherwise the best choice is 24 cores;

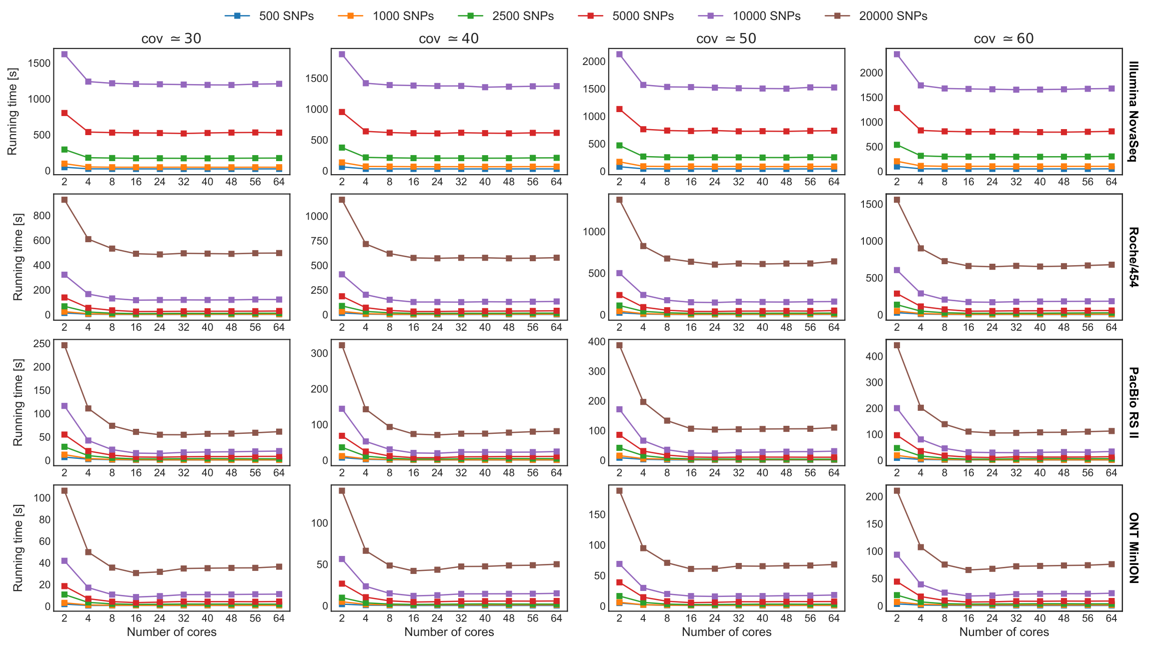


Fig. 6.7 Comparison of the running time required by GenHap on sequencing data generated by four sequencing technologies (Illumina NovaSeq, Roche/454, PacBio RS II, ONT MinION) by varying the coverage values and the number of SNPs. Note that the instances generated using the Illumina NovaSeq technology and characterized by #SNPs = 20000 required more RAM than the amount of memory available on the computing nodes used for the tests. The tests were executed by increasing the number of cores exploited to run GenHap, to evaluate the scalability of the implementation based on distributed computing.

- PacBio RS II: in every test, the fastest executions are generally obtained by exploiting 24 cores to parallelize the GA instances, except when #SNPs = 500 is taken into account. In this case, the running time decreases when 16 cores are exploited to effectively distribute the GA instances on multiple cores. Since the reads generated by relying on this technology have a low accuracy (approximately 87%), which makes the problem more difficult to be solved (i.e., the amount of SNPs to be corrected increases), the scalability of GenHap is emphasized;
- ONT MinION: in all tests, the best choice is 16 cores that allow for efficiently distributing the computational load.

In all tests, a number of cores greater than 24 does not reduce the running time since the overhead introduced by MPI is not entirely mitigated by the required computational load. Furthermore, when the number of sub-problems is lower than the number of available cores, our Master-Slave approach exploits a number of cores equal to the number of sub-problems. On the one hand, when technologies producing short reads are considered, the number of

Table 6.5 Results obtained by GenHap on NovaSeq and MinION datasets, by considering cov $\in \{\sim 30\times, \sim 60\times, \sim 90\times\}$ and #SNPs $\in \{500, 1000, 5000\}$. The performances were evaluated both in terms of *HE* and number of detected haplotypes blocks (values expressed as average $\pm$ standard deviation, calculated over the 10 model instances for each configuration).

		NovaSeq		MinION	
cov	#SNPs	*HE*	Blocks	*HE*	Blocks
	500	5.06 ± 1.070	195 ± 6	0.02 ± 0.066	1 ± 0
$\sim 30\times$	1000	4.38 ± 1.219	388 ± 9	0.05 ± 0.070	1 ± 0
	5000	4.58 ± 0.515	1940 ± 13	0.15 ± 0.439	1 ± 0
	500	4.40 ± 1.139	185 ± 6	0.06 ± 0.098	1 ± 0
$\sim 60\times$	1000	3.96 ± 1.091	382 ± 12	0.00 ± 0.000	1 ± 0
	5000	4.66 ± 0.368	1862 ± 21	0.01 ± 0.010	1 ± 0
	500	4.72 ± 1.669	187 ± 4	0.00 ± 0.000	1 ± 0
$\sim 90\times$	1000	3.93 ± 1.424	370 ± 9	0.00 ± 0.000	1 ± 0
	5000	4.92 ± 0.544	1843 ± 28	0.02 ± 0.025	1 ± 0

haplotype blocks increases along with #SNPs. Since these blocks are solved sequentially and are generally characterized by a number of sub-problems lower than the available cores, 16 or 24 cores allow for balancing the computational load. On the other hand, technologies producing long reads generate a small number of reads that lead to a low number of sub-problems to be solved. Notice that exploiting the accuracy of the reads produced using Illumina NovaSeq, Roche/454 and ONT MinION, the GA instances have a fast convergence to the optimal solutions requiring only a dozen of generations.

In order to quantitatively evaluate how the read length affects the number of haplotype blocks, we generated a set of synthetic (yet realistic) instances by using NovaSeq (Illumina Inc., San Diego, CA, USA) [350] and Oxford Nanopore MinION [212] sequencing platforms. For both sequencing technologies, we generated different instances to collect statistically sound results varying the following parameters: (*i*) #SNPs $\in \{500, 1000, 5000\}$; (*ii*) cov $\in \{\sim 30\times, \sim 60\times, \sim 90\times\}$; (*iii*) average $f_{\text{SNPs}} = 200$. To be more precise, 10 different instances were generated for each combination of cov and #SNPs. The results are evaluated considering the *HE*. As shown in Table 6.5, GenHap obtains better results when applied to infer the pair of haplotypes on the MinION instances, obtaining *HE* values always lower than 0.2%, with a neglectable standard deviation (less than 0.5%). Notice that these instances are always characterized by a single haplotype block. Analyzing the results obtained by running GenHap on the instances generated by the NovaSeq sequencer, the *HE* increases, ranging from 3.93% to 5.06% with a standard deviation up to 1.7%. Due to the short reads characterizing this sequencing platform (i.e., $\sim$150bp), a high number of haplotype blocks is produced. This number increases along with the #SNPs, reaching $\sim$2000 blocks when #SNPs $= 5000$ are analyzed. As shown in Section 6.3.1, the coverage is generally capable of mitigating this problem: indeed, increasing the coverage allows for decreasing the number of haplotypes blocks affecting the instances generated by this technology.

6.4 Conclusions

We presented GenHap, a novel computational method based on GAs to solve the haplotyping problem, which is one of the hot topics in Computational Biology and Bioinformatics. The performance of GenHap was evaluated by considering synthetic (yet realistic) read datasets. The solutions yielded by GenHap are accurate, independently of the number, frequency and coverage of SNPs in the input instances, and without any *a priori* hypothesis about the sequencing error distribution in the reads. Differently from the other state-of-the-art algorithms, GenHap was designed for taking into account datasets produced by the third-generation sequencing technologies, characterized by longer reads and higher coverages with respect to the previous generations. As a matter of fact, the experimental findings show that GenHap works better with the datasets produced by third-generation sequencers. The read accuracy achieved by novel sequencing technologies, such as PacBio RS II and Oxford Nanopore MinION, may be useful for several practical applications. In the case of SNP detection and haplotype phasing in human samples, besides read accuracy, a high-coverage is required to reduce possible errors due to few reads that convey conflicting information [214]. In [398], the authors argued that an average coverage higher than $30\times$ is the *de facto* standard. As a matter of fact, the first human genome that was sequenced using Illumina short-read technology showed that, although almost all homozygous SNPs are detected at a $15\times$ average coverage, an average depth of $33\times$ is required to detect the same proportion of heterozygous SNPs.

We showed that GenHap is capable of outperforminhg HapCol [342], achieving approximately a $4\times$ speed-up in the case of Roche/454 instances, and up to $20\times$ speed-up in the case of the PacBio RS II dataset. Notice that in order to keep the running time constant when the number of SNPs increases, the number of available cores should increase proportionally with #SNPs. Although several approaches have been proposed in literature to solve the haplotyping problem [333, 342], GenHap can be easily adapted to exploit Hi-C data characterized by very high-coverages (up to $90\times$), in combination with other sequencing methods for long-range haplotype phasing [34]. Moreover, GenHap can be also extended to compute haplotypes in organisms with different ploidity [5, 39]. Differently from diploid organisms having two copies of each chromosome set, polyploid organisms have multiple copies of their chromosome sets. Polyploidy has gained scientific interest in the study of the ongoing species diversification phenomena [327]. This characteristic is mainly present in plant genomes, but also in animals (such as salmonid fishes and African clawed frogs) [364]. In these comparative genomic studies, haplotype-aware assemblies play a crucial role in elucidating genetic and epigenetic regulatory evolutionary aspects. Unfortunately, the computational burden of the Haplotype Assembly problem is emphasized in the case of polyploid haplotypes with respect to diploids [103]. Therefore, High-Performance Computing represents a key element for efficient, accurate, and scalable methods for Haplotype Assembly of both diploid and polyploid organisms. Worthy of notice, GenHap could be easily reformulated to consider a multi-objective fitness function (e.g., by exploiting an approach similar to NSGA-III [108]). In this context, a possible future extension of GenHap would consist in introducing other objectives in the fitness function, such as the methylation patterns of the different chromosomes [174] or the gene proximity in maps achieved through Chromosome Conformation Capture (3C) experiments [286].

Chapter 7

Evolutionary method for the analysis of medical images

In this chapter, a novel image enhancement technique based on Genetic Algorithms (GAs) (see Section 2.2.5), called MedGA [379], is described. MedGA specifically aimed at strengthening the sub-distributions in medical images with an underlying bimodal histogram of the gray level intensities. Among the existing Computational Intelligence methods for global optimization, GAs represent the most suitable technique because of the discrete structure of the candidate solutions and the intrinsic combinatorial structure of the problem under investigation.

It is worth noting that all works mentioned in Section 1.5.1 are focused on consumer electronics or medical applications, to obtain more "visually pleasant" images by mainly increasing the contrast of the whole image. On the contrary, the main key novelty of MedGA consists in better revealing the two underlying sub-distributions occurring in an image sub-region characterized by a roughly bimodal histogram, overcoming the limitations of the state-of-the-art contrast enhancement methods, which could produce false edges and consequently over-segmentation when the input images are affected by noise, as in the case of Contrast-Enhanced (CE) Magnetic Resonance Imaging (MRI) data [144]. There exist other algorithms, like Histogram Specification (HS), whose aim is similar to MedGA and consists in matching the histogram of the gray level intensities of the input MR image with a desired output histogram [165]. Nevertheless, this approach cannot be applied to process image datasets characterized by a high variability in gray level distributions, since the histogram to be matched should be defined either *a priori* for the whole dataset, or interactively for each processed image, through a procedure that consists in strengthening and shaping the two underlying sub-distributions.

Even though MedGA exploits the same encoding of candidate solutions defined in [188] and [117], its purpose is very different since it was designed to explicitly strengthen the two sub-distributions of medical images characterized by an underlying bimodal histogram. To this aim, we defined a specific fitness function that emphasizes the two Gaussian distributions composing a bimodal histogram. This achievement plays a fundamental role for threshold-based segmentation approaches, since they strongly rely on the assumption that the bimodal histogram under investigation is composed of two nearly Gaussian distributions with almost equal size and variance [477].

Finally, MedGA also differs from the approaches based on Genetic Programming (GP) whose generated solutions might have a large size [69], even when the GP model is implemented efficiently, thus representing a limitation that could significantly impair the readability and interpretability of the final outcome. Moreover, MedGA does not require any user interaction step, differently to [343] where the user, being directly involved in the tournament selection, controls the evolution of simple programs that enhance and integrate multiple gray-scale images into a single pseudo-color image. Therefore, the selection is based on the output image (i.e., phenotype) rather than the structure and size of the program (i.e., genotype) [343].

We also integrated MedGA as a pre-processing stage into a novel framework for image enhancement, automatic global thresholding and segmentation, which has been applied to different clinical scenarios involving bimodal MR image analysis [377]. We remind that medical image segmentation concerns both detection and delineation of anatomical or physiological structures from the background, distinguishing among the different components included in the image [331]. This important task allows for the extraction of clinically useful information and features in medical image analysis [458, 218]. Accordingly, computer-assisted approaches enable quantitative imaging [122], whose aim is to derive accurate and objective measurements from digital images regarding a Region of Interest (ROI) [482, 160]. Indeed, image segmentation is still one of the most compelling research areas, especially in medical image analysis [122]. Accurately delineating the ROIs is a critical task, since manual segmentation procedures are time-expensive, error-prone, and operator-dependent (i.e., they do not guarantee results repeatability).

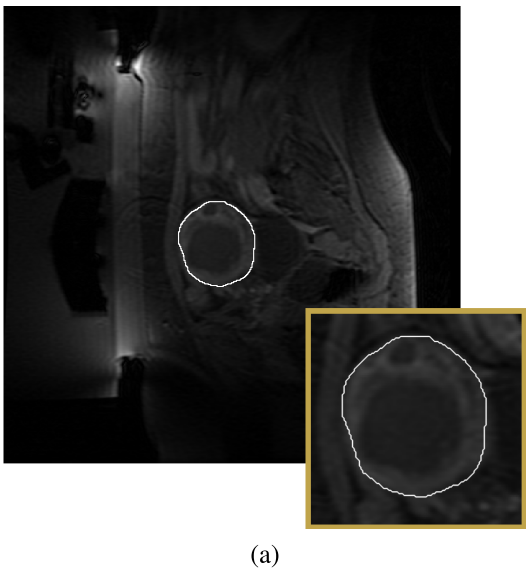
7.1 Image thresholding

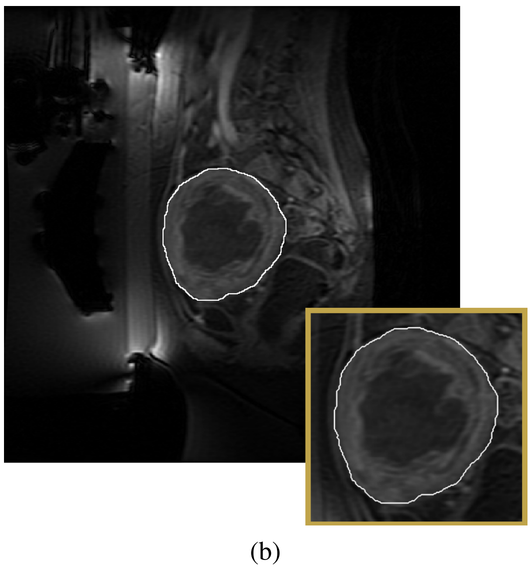
The most straightforward unsupervised Pattern Recognition technique for automatic image segmentation is global thresholding, which generally consists in classifying pixels according to fixed criteria, usually specified as ranges of intensities [365]. In particular, binarization is a segmentation technique that partitions the input image into two classes by considering a certain intensity threshold value θ. Despite its simplicity, this strategy provides an efficient and effective segmentation technique, according to the different intensities in the foreground and background regions of an image. The threshold value θ must be carefully chosen, considering the features of the image underlying the pixel intensity values. Consequently, given an image I consisting of $M \times N$ pixels, θ defines two different classes, by dividing the histogram of the gray levels $\mathcal{H}$ into two parts, namely $\mathcal{H}_1$ and $\mathcal{H}_2$. The pixels in the image I are partitioned into the two sub-regions $R_1 = \{I(x,y) : I(x,y) > \theta\}$ and $R_2 = \{I(x,y) : I(x,y) \leq \theta\}$, for every $x = 1,\ldots,M$ and $y = 1,\ldots,N$.

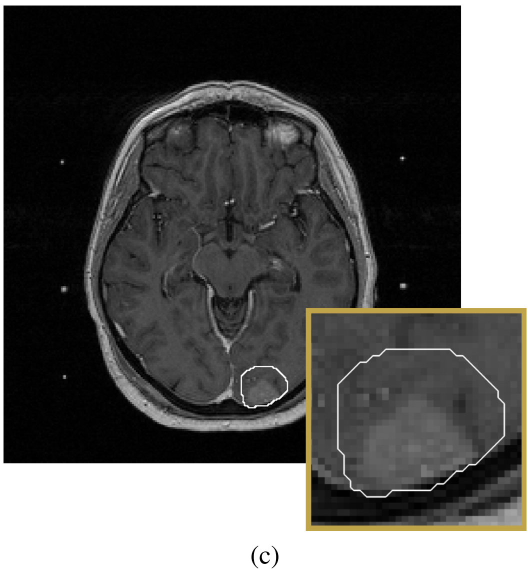
Several literature methods have been proposed to implement adaptive thresholding methods, able to automatically select a proper value for each analyzed image. The most widespread algorithms for dynamic thresholding are: the Iterative Optimal Threshold Selection (IOTS) [359]; the method proposed by Otsu [326]; the Minimum-Error Thresholding (MET) method conceived by Kittler and Illingworth [230], later extended by Ye and Danielsson [483]. All these approaches are closely related and strongly rely on images characterized by bimodal histograms (see, e.g., [477]). In addition, the two populations (i.e., foreground and background pixels in the case of two-class image segmentation), assumed to be nearly Gaussian distributions, should be characterized by approximately equal size and variance [241]. When these assumptions are not satisfied—i.e., the pixel intensity distribution is not approximately bimodal—the aforementioned algorithms show some limitations. As a matter of fact, the optimal threshold θ_{opt}—especially in the case of Otsu's method—either over- or under-estimates the ROI, since the computed threshold tends to split the class with larger size and to bias towards the class with larger variance. Under these conditions, the IOTS method [359, 445] could provide better results than Otsu's method [326] when the sizes of the two classes are highly different [476]. In addition, Medina-Carnicer *et al.* in [282] showed that these algorithms often perform poorly with unimodal distributions of gray levels. Moreover, in the case of images affected by intensity overlap, the IOTS algorithm is less likely to either over- or under-estimate the threshold, when compared to other techniques selecting a threshold between the two peaks of the histogram, even if the histogram is not strongly bimodal [406], in particular when applied to medical images [242].

7.2 MedGA

MedGA is a global enhancement technique able to improve the details of medical images characterized by an underlying bimodal histogram. Given a medical image wherein a ROI needs to be enhanced to achieve further analyses, MedGA aims at improving the ROI quality







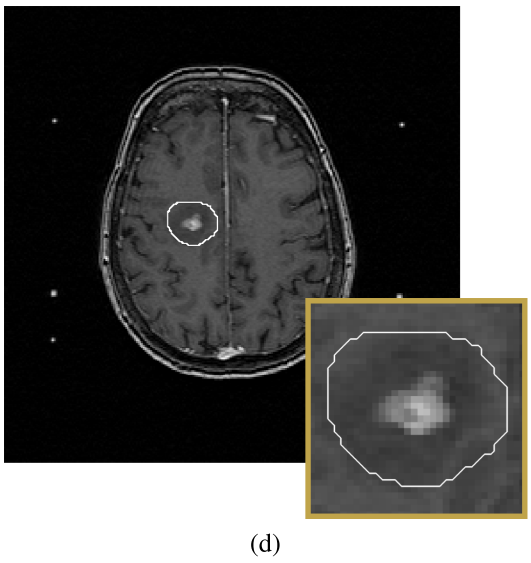


Fig. 7.1 Examples of input MR images: (a, b) uterine fibroid inside the uterus region; (c, d) brain tumor inside a ROI bounding region selected by the healthcare operator. The image regions including the ROIs, defined by the white contour and zoomed at the bottom right of each sub-figure, are characterized by nearly bimodal histograms.

to facilitate the classification among different neighboring tissues, in order to support both the interpretation tasks by experienced radiologists and automated image analysis approaches.

The image enhancement carried out by MedGA focuses on the pixels within a sub-region of the input MRI, called ROI bounding region, including the ROI itself. To be more precise, starting from an MR image (Figure 7.1), a bounding region that roughly includes the actual ROI (e.g., the uterus region including uterine fibroids in Figure 7.1a and 7.1b) is identified by using either a manual or a computational method. Afterwards, the entire original MR image is cropped at the smallest rectangular box including the previously identified ROI bounding region. The pixels included in the rectangular cropped image, but external to the ROI bounding region, are set to zero (i.e., the black level). By so doing, an image characterized by a nearly bimodal histogram is obtained. Then, a linear contrast stretching is applied to the initial full range of gray levels, that is, the ordered set $\mathcal{L}_{in} = [l_{in}^{(min)}, l_{in}^{(min)}+1, \ldots, l_{in}^{(max)}-1, l_{in}^{(max)}] \subset \mathbb{N}$, where $l \neq l'$ for any $l, l' \in \mathcal{L}_{in}$. The integers $l_{in}^{(min)}$ and $l_{in}^{(max)}$ in $\mathcal{L}_{in}$ denote the minimum and maximum non-zero gray levels of the analyzed image sub-region, respectively. The linear contrast stretching applied to $\mathcal{L}_{in}$ exploits the extended range of the non-zero gray levels, that is, the ordered set $\mathcal{L}'_{in} = [1, \ldots, l_{in}^{(max)}] \subset \mathbb{N}$, where typically $l_{in}^{(min)} > 1$. Note that the zero-padding pixels (i.e., the pixels corresponding to the level 0) are not taken into account, and that any element of $\mathcal{L}_{in}$ is also an element of $\mathcal{L}'_{in}$. This normalization operation, which employs only values of gray levels already representable in the initial dynamic range, does not alter the image content and allows MedGA to process additional intensity levels with respect to the initial full range $\mathcal{L}_{in}$, by considering the intensity variability within the analyzed MRI dataset. It is worth noting that the pre-processing described hereby does not exploit any method that could affect the actual pictorial content (e.g., spatial or frequency filtering).

7.2.1 Implementation strategy

MedGA exploits a population P of individuals $C_i = [C_i(1), C_i(2), \ldots, C_i(n)]$ (with $i = 1, \ldots, |P|$) defined as circular arrays of integer numbers of size n, where $n = |\mathcal{L}'_{in}|$ corresponds to the number of different gray levels belonging to $\mathcal{L}'_{in}$ identified in the input MR image (i.e., the gray levels whose frequency is greater than zero in the input MR image). Each individual $C_i \in P$ is randomly initialized by sampling n integer values from the discrete uniform distribution in $\mathcal{L}'_{in}$. The n values are then sorted in ascending order so that the intensity levels $C_i(j)$ (with $j = 1, \ldots, n$) codified by the individual can be mapped to the intensity levels of the input MR image (i.e., the gray level frequencies of the input MR image are assigned to the corresponding intensity levels of the individual). During the initialization of the individuals, if an integer value is sampled more than once then the frequency values of the input MR image, corresponding to these gray level intensities, are summed up and assigned to the same gray level of the individual.

The rationale of MedGA is to process the ordered set $\mathcal{L}'_{in}$ by modifying its gray levels using a sequence of genetic operators, to obtain a solution characterized by a stronger bimodal gray level distribution in an output gray-scale range $\mathcal{L}_{out} = [l_{out}^{(min)}, \ldots, l_{out}^{(max)}] \subset \mathbb{N}$, where any element of $\mathcal{L}_{out}$ is also an element of $\mathcal{L}'_{in}$. In such a way, a direct mapping between the gray levels of the original image and the final one is defined, so that each gray level in the original histogram is replaced with the gray level value contained by the same position in the final best solution $C_{best} \in P$. MedGA realizes this global intensity transformation by improving the separation between $\mathcal{H}_{1,i}$ and $\mathcal{H}_{2,i}$, which represent the dark and bright sub-regions of the histogram $\mathcal{H}_i$ encoded by the individual C_i, respectively. To this aim, MedGA exploits the optimal threshold $\theta_{opt,i}$ adaptively selected using the IOTS method [359, 445]. This procedure yields enhanced medical images that better reveal the bimodal intensity distribution in computer-assisted ROI extraction tasks.

At each iteration of MedGA, a number of individuals properly selected from the current population are inserted into intermediate populations, and modified by means of crossover and mutation operators. Note that, at each iteration, each individual C_i belonging to the current population codifies for an ordered set $\mathcal{L}_{out,i} = [C_i(1), \ldots, C_i(n)] = [l_{out,i}^{(min)}, \ldots, l_{out,i}^{(max)}]$, which represents a modified gray level distribution of $\mathcal{L}'_{in}$. In order to simplify the notation, in what follows we do not explicitly express that, *at each iteration*, each individual C_i is (possibly) represented by a different circular array corresponding to $\mathcal{L}_{out}$.

For the selection of individuals, MedGA exploits a *tournament* strategy for three main reasons: (*i*) the selection pressure can be controlled by setting the tournament size k (with $k \ll |P|$); (*ii*) the fitness evaluations are performed only on the k individuals selected for the tournaments, and not on the whole population; (*iii*) this technique could be easily implemented on parallel architectures [293].

As shown in Figure 7.2, a *single point crossover* operator is applied with a given probability p_c to the individuals selected by the tournament strategy and belonging to the first intermediate population P'. Namely, given two parent individuals $\text{Parent}_1, \text{Parent}_2 \in P'$:

1. a crossover point c_p is randomly selected from the ordered set $[1, 2, \ldots, n]$;

2. if $c_p > \text{round}(\frac{n}{2})$, the first offspring is generated using the genes $1, \ldots, c_p - h_p - 1$ and $c_p, \ldots, n$ from Parent_1, and the genes $c_p - h_p, \ldots, c_p - 1$ from Parent_2. Otherwise, it is generated using the genes $c_p, \ldots, c_p + h_p - 1$ from Parent_1, and the genes $1, \ldots, c_p - 1$ and $c_p + h_p, \ldots, n$ from Parent_2;

3. the second offspring is generated changing the Parent_1's genes with those of Parent_2 and viceversa.

The two offspring generated by swapping 50% of the values between the two parents are then inserted into a second intermediate population P''.

The *mutation* operator is applied with probability p_m to each element $C_i(j) \in C_i = [l_{out,i}^{(min)}, \ldots, l_{out,i}^{(max)}]$ of each individual belonging to P'', where $l_{out,i}^{(min)}$ and $l_{out,i}^{(max)}$ are the mini-

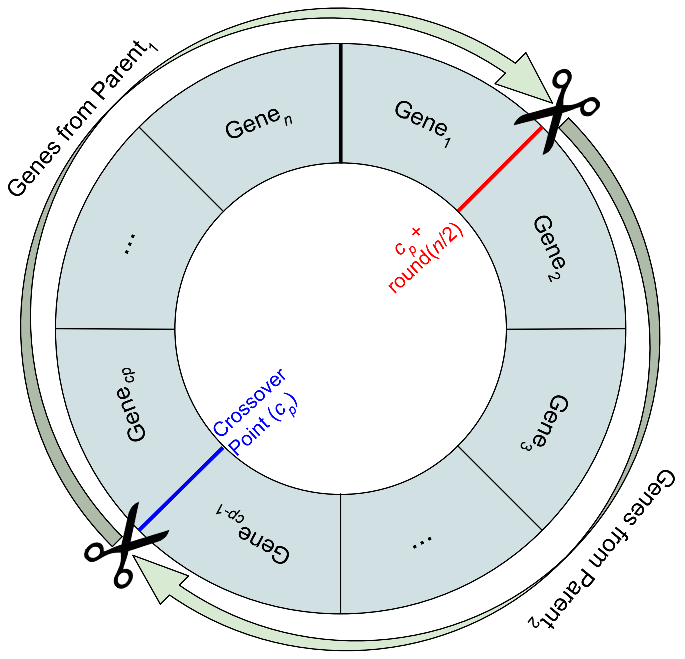


Fig. 7.2 Representation of the crossover strategy used for generating the first offspring from two parents Parent$_1$ and Parent$_2$. The second offspring is generated changing the Parent$_1$'s genes with those of Parent$_2$ and viceversa.

mum and maximum non-zero gray levels encoded by C_i during the current iteration, respectively. In particular, if the gray level intensity encoded in $C_i(j)$ is smaller than the optimal threshold $\theta_{opt,i}$ evaluated by IOTS at that iteration for the individual C_i, then an integer is randomly sampled from the uniform distribution in $[l_{out,i}^{(min)}, \dots, \theta_{opt,i} - 1] \subset \mathbb{N}$ to update the value $C_i(j)$; otherwise, an integer is randomly sampled from the uniform distribution in $[\theta_{opt,i}, \dots, l_{out,i}^{(max)}] \subset \mathbb{N}$ to update the value $C_i(j)$.

Finally, to prevent the quality of the best solution from decreasing during the optimization, MedGA also exploits an *elitism* strategy, so that the best individual from the current population is copied into the next population without undergoing the genetic operators.

Figure 7.3 illustrates the overall procedure of MedGA, by presenting the initialization phase as well as the flow diagram of the proposed GA for image enhancement. The final best solution shows the achieved result on MRI data characterized by a bimodal histogram, emphasizing the two underlying distributions for the subsequent image thresholding phase, according to the optimal adaptive threshold θ_{opt}, computed on the final best solution C_{best}, by using the simple IOTS algorithm [359, 445].

To evaluate the quality of the individuals throughout the optimization, the fitness function has been conceived to obtain a bimodal histogram in the gray levels intensities, therefore facilitating further automated image processing phases. The fitness function $\mathcal{F}(\cdot)$ used by

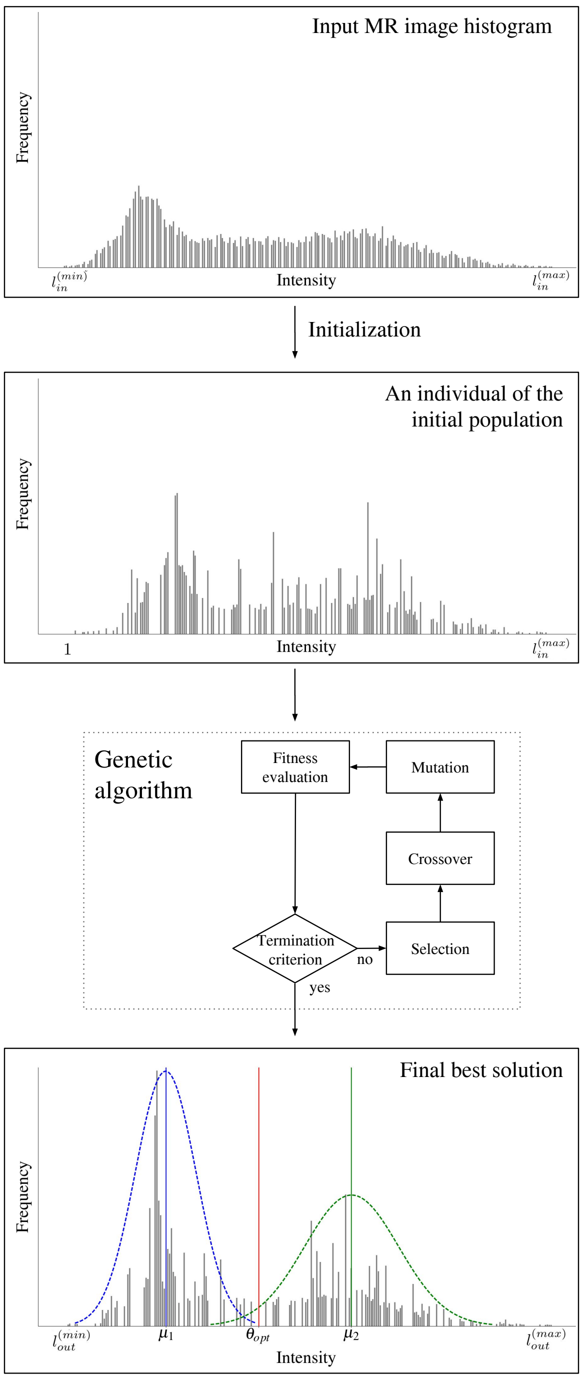


Fig. 7.3 Workflow of MedGA: the individuals are initialized according to the characteristics of the input MR image, and processed by the GA. The final best solution of MedGA strengthens the two underlying distributions in the gray levels intensity characterized by mean values μ_1 and μ_2 and standard deviations σ_1 and σ_2, respectively. The two distributions are highlighted in the plot with blue and green dashed lines.

MedGA fosters individuals C_i characterized by a histogram with two well separated normal distributions, having an equal distance from the optimal threshold $\theta_{opt,i}$. To this purpose, at each iteration MedGA estimates, by using the efficient IOTS algorithm [359, 445], the mean values $\mu_{1,i}$ and $\mu_{2,i}$ of the two components $\mathcal{H}_{1,i}$ and $\mathcal{H}_{2,i}$ of the histogram $\mathcal{H}_i$, encoded by each individual C_i, according to the current optimal threshold $\theta_{opt,i}$. Afterwards, at each iteration the fitness function of every individual C_i is calculated as follows:

$$\begin{aligned}
\mathcal{F}(C_i) &= \tau_1 + \tau_2 + \tau_3, \quad \text{where:} \\
\tau_1 &= \left|2 \cdot \theta_{opt,i} - \mu_{1,i} - \mu_{2,i}\right|, \\
\tau_2 &= \left|\omega_{1,i} - 3\sigma_{1,i}\right|, \\
\tau_3 &= \left|\omega_{2,i} - 3\sigma_{2,i}\right|.
\end{aligned} \tag{7.1}$$

The terms $\omega_{1,i} = \frac{1}{2}(\theta_{opt,i} - \min_{j \in \{1,\dots,n\}} \{C_i(j)\})$ and $\omega_{2,i} = \frac{1}{2}(\max_{j \in \{1,\dots,n\}} \{C_i(j)\} - \theta_{opt,i})$ correspond to the half width of $\mathcal{H}_{1,i}$ and $\mathcal{H}_{2,i}$, respectively, while $\sigma_{1,i}$ and $\sigma_{2,i}$ are the standard deviations of $\mathcal{H}_{1,i}$ and $\mathcal{H}_{2,i}$, respectively. The three terms of the fitness function $\mathcal{F}(\cdot)$ cooperate together to achieve the desired image enhancement: τ_1 aims at maintaining the mean values $\mu_{1,i}$ and $\mu_{2,i}$ equidistant from the yielded optimal threshold $\theta_{opt,i}$, while τ_2 and τ_3 are designed to force the sub-histograms $\mathcal{H}_{1,i}$ and $\mathcal{H}_{2,i}$ to approximate normal distributions.

We exploit the empirical property of normal distributions related to the coverage probability with respect to the standard deviation. To be more precise, we consider the so-called 3-σ rule, which states that approximately 99.73% of the values lie within 3σ according to: $Pr(\mu - 3\sigma \leq X \leq \mu + 3\sigma) \approx 0.9973$, where μ, σ and X represent the mean, the standard deviation and an observation from a normally distributed random variable, respectively.

Two examples of image enhancement results, achieved by MedGA on a uterine fibroid and on a brain tumor, are shown in Figures 7.4 and 7.5, respectively. In the case of uterine fibroids, MedGA enhances the input MR image by making fibroid regions more uniform and with sharper edges in terms of both visual human perception and automated image segmentation. The histogram in Figure 7.4d points out that the output image is characterized by a more defined bimodal distribution compared to the initial image (Figure 7.4b), which presents approximately a trimodal gray level distribution. In the case of brain tumors, MedGA enhances the underlying bimodal distribution related to contrast-enhancing tumoral tissue and brain healthy tissues on CE-MR images. This visual achievement is endorsed by the histogram of the enhanced image (see Figure 7.5d) that shows two more distinct peaks with respect to the initial gray level distribution (see Figure 7.5b).

It is worth noting that, in the case of further automated analysis, ROI pixel classification can be carried out by means of a basic threshold-based segmentation approach, since MR

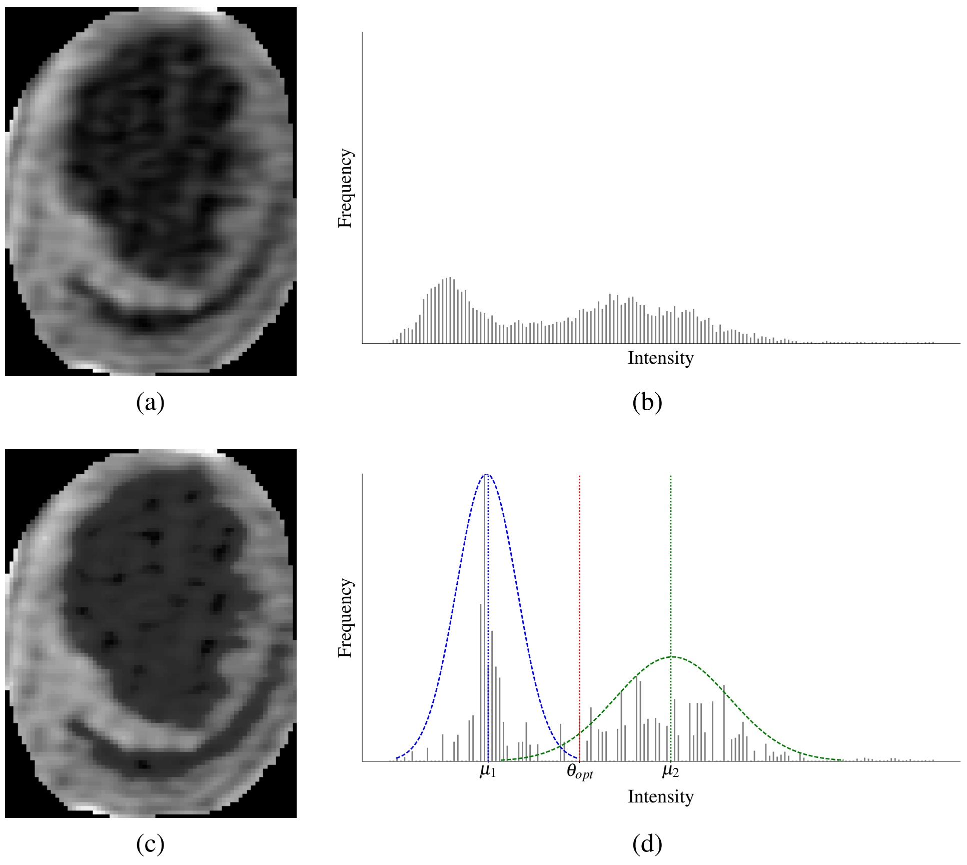


Fig. 7.4 Enhanced image obtained by MedGA on an example of uterine fibroid (size: 89×70 pixels): (a) normalized input image using linear contrast stretching on the initial full range of the masked MR image; (c) resulting image after the application of the pre-processing using MedGA. The histograms corresponding to the sub-images in (a) and (c) are shown in (b) and (d), respectively. The final histogram emphasizes the two underlying distributions in the gray levels intensity characterized by mean values μ_1 and μ_2, and standard deviations σ_1 and σ_2, respectively. The two distributions are highlighted with blue and green dashed lines.

images enhanced with MedGA reveal a more precise separation between the two (possibly overlapping) sub-distributions in the histogram. Accordingly, MedGA allows for dealing with image histograms that do not meet the assumptions imposed by thresholding techniques, regarding bimodal histograms composed of two nearly Gaussian distributions with almost equal size and variance [477]. Indeed, MedGA enhances the image thresholding results on MRI data, as shown in Figure 7.3, where the final best solution improves the underlying bimodal nature of the input histogram.

A sequential and a parallel version of MedGA have been implemented. The sequential version has been entirely developed using the `Python` programming language (version 2.7.12), while the parallel version is based on a Master-Slave paradigm employing `mpi4py`,

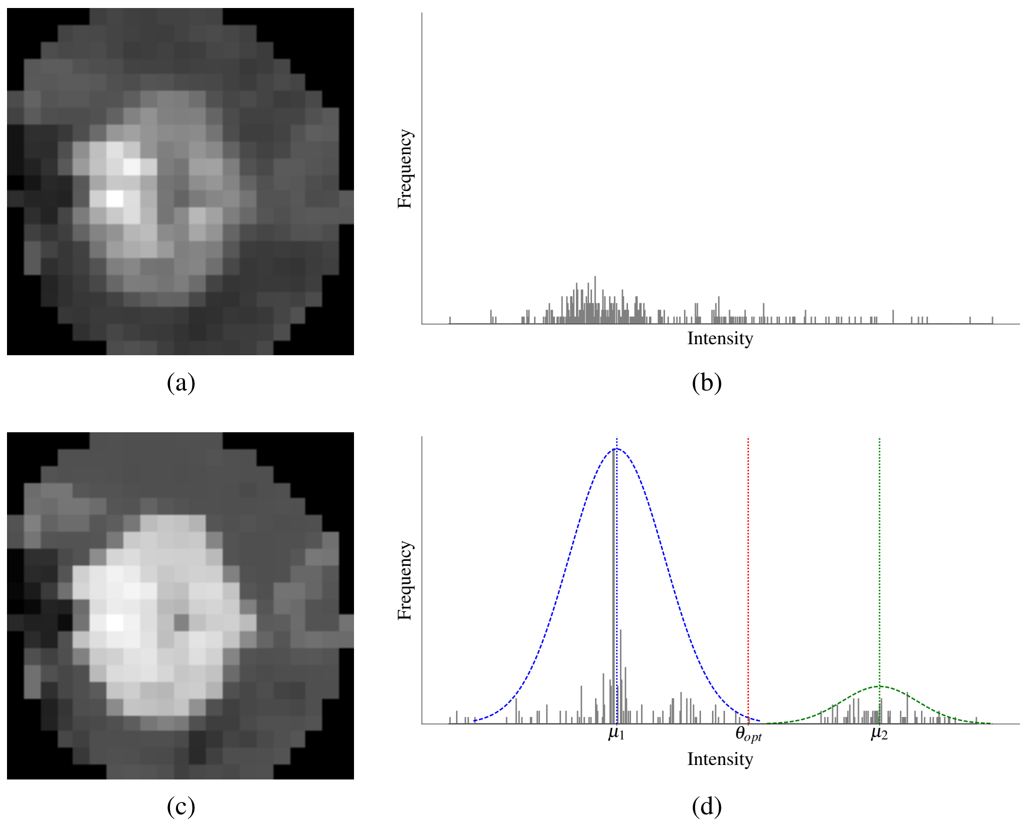


Fig. 7.5 Enhanced image obtained by MedGA on an example of brain tumor (size: 21×21 pixels): (a) normalized input image using linear contrast stretching on the initial full range of the masked MR image; (c) resulting image after the application of the pre-processing using MedGA. The histograms corresponding to the sub-images in (a) and (c) are shown in (b) and (d), respectively. The final histogram emphasizes the two underlying distributions in the gray levels intensity characterized by mean values μ_1 and μ_2 and standard deviations σ_1 and σ_2, respectively. The two distributions are highlighted in the plot with blue and green dashed lines.

which provides bindings of the Massage Passing Interface (MPI) specifications for `Python` to leverage High-Performance Computing (HPC) resources [98].

7.3 Evaluation metrics and data

7.3.1 Image enhancement metrics

In this section, we recall the definition of the metrics typically used to evaluate image enhancement approaches, which will be exploited to assess the performance of MedGA. These metrics are essential to quantitatively evaluate the effects of image enhancement techniques, since measuring the "quality" of an image might be strongly subjective. In particular, we benefit here from the metrics considered in [188] to assess the capability of image enhancement approaches in improving contrast, details and human visual perception.

Let I_{orig} and I_{enh} be the original input image and the enhanced image, respectively, consisting in M rows and N columns. Considering that the range of gray levels of the output image $\mathcal{L}_{out} = [l_{out}^{(min)}, \ldots, l_{out}^{(max)}]$ could be different from the original range $\mathcal{L}_{in} = [l_{in}^{(min)}, \ldots, l_{in}^{(max)}]$, as a first step before computing the metrics we remap the output pixel intensities (i.e., the gray levels) into the original range, as follows:

$$\tilde{I}_{enh}(a,b) = \frac{(I_{enh}(a,b) - l_{out}^{(min)}) \cdot (l_{in}^{(max)} - l_{in}^{(min)})}{(l_{out}^{(max)} - l_{out}^{(min)})} + l_{in}^{(min)}, \tag{7.2}$$

where $a = 1, \ldots, M$ and $b = 1, \ldots, N$. Note that we actually consider the extended range $\mathcal{L}'_{in}$ for I_{orig}, as explained in Section 7.2.

The Peak Signal-to-Noise Ratio (*PSNR*) denotes the ratio between the maximum possible intensity value of a signal and the distortion between the input and output images:

$$PSNR = 10 \cdot \log_{10}\left(\frac{(l_{in}^{(max)})^2}{MSE}\right) = 20 \cdot \log_{10}\left(\frac{l_{in}^{(max)}}{\sqrt{MSE}}\right), \tag{7.3}$$

where $MSE = \frac{1}{M \times N} \sum_{a=1}^{M} \sum_{b=1}^{N} \left\| I_{orig}(a,b) - \tilde{I}_{enh}(a,b) \right\|^2$ is the Mean Squared Error (MSE), which allows us to compare the pixel values of I_{orig} to those of $\tilde{I}_{enh}$.

Furthermore, the *PSNR* is usually expressed in terms of the logarithmic Decibel scale. With regard to our application, we employ only a limited portion of the full dynamic range of the 16-bit images (see Section 7.2); we thus use as the largest possible value the maximum intensity value present in the original image (i.e., $l_{in}^{(max)} = \max\{\mathcal{L}_{in}\} = \max\{\mathcal{L}'_{in}\}$) instead of the maximum representable value in a 16-bit image (i.e., $2^{16} - 1 = 65,535$).

In [297], the authors stated that good contrast and enhanced images are characterized by high numbers of *edgels* (i.e., pixels belonging to an edge), and that an enhanced image should have a higher intensity of the edges, compared to its non-enhanced counterpart [381]. Therefore, a good enhancement technique should yield satisfactory results in the case of standard vision processing tasks, such as segmentation or edge detection [411]. Here, to evaluate image enhancement of MRI data, we employ the method proposed by [63], which is a highly reliable and mathematically well-defined edge detector. This approach deals with weak edges and accurately determines edgels, by applying a double threshold (to identify potential edges) and a hysteresis-based edge tracking. Let $\mathcal{M}_{Canny}$ be the edge map yielded by the Canny's edge detector, which is a binary image wherein only edgels are set to 1. The number of detected edges (#DE) in $\mathcal{M}_{Canny}$ is computed as:

$$\#\text{DE} = \sum_{a=1}^{M} \sum_{b=1}^{N} \mathcal{M}_{Canny}(a,b). \tag{7.4}$$

An additional metrics, called Absolute Mean Brightness Error (*AMBE*) [80, 20], can be employed to measure the brightness preservation of the enhanced image:

$$AMBE = \frac{\left|\mathbb{E}[I_{orig}] - \mathbb{E}[\tilde{I}_{enh}]\right|}{L}, \tag{7.5}$$

where $\mathbb{E}[\cdot]$ denotes the expected (mean) value of a gray level distribution. *AMBE* is normalized in $[0,1]$, divided by $L = l_{in}^{(max)} - l_{in}^{(min)}$, which is the dynamic range of the input gray-scale (in our case, $\mathcal{L}'_{in}$). Note that low values of *AMBE* denote that the mean brightness of the original image is preserved.

Finally, we consider an alternative quality metrics called Structural Similarity Index (*SSIM*) [463], used to assess the image degradation perceived as variations in structural information [47]. The structural information defines the attributes that represent the structure of objects in the image, independently of the average luminance and contrast. In particular, local luminance and contrast are taken into account since overall values of luminance and contrast can remarkably vary across the whole image. *SSIM* is based on the degradation of structural information—assuming that human visual perception is highly adapted for extracting structural information from a scene—and compares local patterns of pixel intensities. As a matter of fact, natural image signals are highly structured, since pixels are strongly dependent on each other, especially those close by. These dependencies convey important information about the structure of the objects in the viewing field. Let $\mathbf{X}$ and $\mathbf{Y}$ be the I_{orig} and I_{enh} image signals, respectively; *SSIM* combines three relatively independent terms:

- the luminance comparison $l(\mathbf{X},\mathbf{Y}) = \frac{2\mu_{\mathbf{X}}\mu_{\mathbf{Y}}+\kappa_1}{\mu_{\mathbf{X}}^2+\mu_{\mathbf{Y}}^2+\kappa_1}$;
- the contrast comparison $c(\mathbf{X},\mathbf{Y}) = \frac{2\sigma_{\mathbf{X}}\sigma_{\mathbf{Y}}+\kappa_2}{\sigma_{\mathbf{X}}^2+\sigma_{\mathbf{Y}}^2+\kappa_2}$;
- the structural comparison $s(\mathbf{X},\mathbf{Y}) = \frac{\sigma_{\mathbf{XY}}+\kappa_3}{\sigma_{\mathbf{X}}\sigma_{\mathbf{Y}}+\kappa_3}$;

where $\mu_{\mathbf{X}}$, $\mu_{\mathbf{Y}}$, $\sigma_{\mathbf{X}}$, $\sigma_{\mathbf{Y}}$, and $\sigma_{\mathbf{XY}}$ are the local means, standard deviations, and cross-covariance for the images $\mathbf{X}$ and $\mathbf{Y}$, while $\kappa_1, \kappa_2, \kappa_3 \in \mathbb{R}^+$ are regularization constants for luminance, contrast, and structural terms, respectively, exploited to avoid instability in the case of image regions characterized by local mean or standard deviation close to zero. Typically, small non-zero values are employed for these constants; according to [463], an appropriate setting is $\kappa_1 = (0.01 \cdot L)^2$, $\kappa_2 = (0.03 \cdot L)^2$, $\kappa_3 = \kappa_2/2$, where L is the dynamic range of the pixel values in I_{orig} (represented in $\mathcal{L}'_{in}$). *SSIM* is then computed by combining the components described above:

$$SSIM = l(\mathbf{X},\mathbf{Y})^{\alpha} \cdot c(\mathbf{X},\mathbf{Y})^{\beta} \cdot s(\mathbf{X},\mathbf{Y})^{\gamma}, \tag{7.6}$$

where α, β, $\gamma > 0$ are weighting exponents. As reported in [463], if $\alpha = \beta = \gamma = 1$ and $\kappa_3 = \kappa_2/2$, the *SSIM* becomes:

$$SSIM = \frac{(2\mu_{\mathbf{X}}\mu_{\mathbf{Y}} + \kappa_1)(2\sigma_{\mathbf{XY}} + \kappa_2)}{\left(\mu_{\mathbf{X}}^2 + \mu_{\mathbf{Y}}^2 + \kappa_1\right)\left(\sigma_{\mathbf{X}}^2 + \sigma_{\mathbf{Y}}^2 + \kappa_2\right)}. \tag{7.7}$$

SSIM generalizes the Universal Quality Index (*UQI*), defined in [462], which is obtained by setting $\kappa_1 = \kappa_2 = 0$, and yields unstable results when either $\left(\mu_{\mathbf{X}}^2 + \mu_{\mathbf{Y}}^2\right)$ or $\left(\sigma_{\mathbf{X}}^2 + \sigma_{\mathbf{Y}}^2\right)$ tends to zero. Notice that the higher the *SSIM* value, the higher the structural similarity, implying that the enhanced image I_{enh} and the original image I_{orig} are quantitatively similar.

7.3.2 Image segmentation metrics

Hereafter, the used spatial overlap-based and distance-based metrics for medical image segmentation evaluation are defined, according to the formulations given in [423]. These evaluation metrics aim at quantifying the accuracy of a computer-assisted segmentation method against a gold-standard delineation, as described in [490].

Spatial overlap-based metrics

In order to perform a fair comparison with respect to the real measurement and to quantify the area or the volume of the segmented object in the treatment phase, it is mandatory to calculate several spatial overlap-based metrics. These metrics are calculated considering the gold-standard regions obtained in a manual way by an expert (R_G), and the automatically segmented regions (R_S) performed by a computational method, in terms of sample sets composed of pixels (2D) or voxels (3D). Let I_{MRI} be the entire input MR image, including all possible segmented regions, and considering the confusion matrix, *true positive (TP)*, *false positive (FP)*, *false negative (FN)*, and *true negative (TN)* regions are defined as: $R_{TP} = R_S \cap R_G$, $R_{FP} = R_S - R_G$, $R_{FN} = R_G - R_S$, and $R_{TN} = I_{\text{MRI}} - R_G - R_S$, respectively. We used the following spatial overlap-based metrics:

- *Dice Similarity Index (or Sørensen-Dice coefficient)* is the most significant measure to estimate the accuracy in medical image segmentation and is highly dependent on the size of the compared regions (the higher the value, the better the segmentation results)

[494]:

$$DSI = \frac{2 \cdot |R_{TP}|}{|R_S| + |R_G|} \cdot 100\%; \tag{7.8}$$

- *Sensitivity*, as known as *True Positive Ratio (TPR)*, detects the correct detection ratio, that is, the area of the automatic segmentation that includes the area of the manually segmented region (high values imply good segmentation results):

$$SEN = TPR = \frac{|R_{TP}|}{|R_{TP}| + |R_{FN}|} \cdot 100\%; \tag{7.9}$$

- *Specificity* quantifies the ability of the automated method to exclude wrong regions in the segmented ROI. This statistical measure is usually defined as *True Negative Ratio (TNR)* in binary classification and indicates the portion of background pixels correctly identified by the automatic segmentation with respect to the gold-standard R_T, according to (7.10):

$$TNR = \frac{|R_{TN}|}{|R_{TN}| + |R_{FP}|} \cdot 100\%. \tag{7.10}$$

  However, this formulation is ineffective when unbalanced data (i.e., the ROI is very small with respect to the whole image) are analyzed. Consequently, we decided to use the definition given in (7.11):

$$SPC = \left(1 - \frac{|R_{FP}|}{|R_S|}\right) \cdot 100\%, \tag{7.11}$$

  where $\frac{|R_{FP}|}{|R_S|}$ is the incorrect fraction of the ROI included by the automatic procedure (high values of *SPC* mean good segmentation result).

Spatial distance-based metrics

Generally, the quality of a segmentation method cannot be evaluated by calculating area-based metrics only. In clinical practice, one of the primary objectives of ROI segmentation is to achieve a precise boundary tracing. In such a case, the distance between the ROI boundaries generated by the automatic approach and those manually contoured by an expert physician (i.e., the gold-standard) must be measured. Accordingly, to take into account the spatial position of the voxels, distance-based metrics are highly recommended. We define $\mathcal{G} = \{\mathbf{g}_a : a = 1, 2, \ldots, A\}$ as the set of vertices composing the ground truth, and $\mathcal{S} = \{\mathbf{s}_b : b = 1, 2, \ldots, B\}$ the set of vertices forming the automatically generated boundaries.

We calculate the distance between an element of the set $\mathcal{S}$, representing a segmented ROI, and the set $\mathcal{G}$ as follows:

$$d(\mathbf{s}_b, \mathcal{G}) = \min_{a \in \{1,2,\ldots,A\}} \|\mathbf{s}_b - \mathbf{g}_a\|, \tag{7.12}$$

where $\|\mathbf{s}_b - \mathbf{g}_a\|$ corresponds to the Euclidean distance between two points in a 2D space.

Now, we can define the main three measures based on the distance of points and independent of region size:

- *Mean Absolute Distance (MAD)* measures the average error between the boundaries of the target ROI and the automatic one in the segmentation process:

$$MAD = \frac{\sum_{b=1}^{B} d(\mathbf{s}_b, \mathcal{G})}{N}; \tag{7.13}$$

- *Maximum absolute Distance (MaxD)* represents the maximum difference between the ROI boundaries of the regions R_A and R_T:

$$MaxD = \max_{b \in \{1,2,\ldots,B\}} \{d(\mathbf{s}_b, \mathcal{G})\}; \tag{7.14}$$

- *Hausdorff Distance (HD)* measures the extent between the set of vertices $\mathcal{S}$ and $\mathcal{G}$, corresponding to the boundaries of regions R_S and R_G, respectively. It is defined as:

$$HD = \max\{h(\mathcal{G}, \mathcal{S}), h(\mathcal{S}, \mathcal{G})\}, \tag{7.15}$$

  where $h(\mathcal{G}, \mathcal{S}) = \max_{\mathbf{g} \in \mathcal{G}} \left\{ \min_{\mathbf{s} \in \mathcal{S}} \{d(\mathbf{g}, \mathbf{s})\} \right\}$, $h(\mathcal{S}, \mathcal{G}) = \max_{\mathbf{s} \in \mathcal{S}} \left\{ \min_{\mathbf{g} \in \mathcal{G}} \{d(\mathbf{s}, \mathbf{g})\} \right\}$, and $d(\mathbf{g}, \mathbf{s})$ is the Euclidean distance in an n-dimensional Euclidean space $\mathbb{R}^n$.

In order to make these metrics independent on the spatial resolution of different datasets (i.e., pixel spacing), they are expressed in pixels. Accordingly, higher values of such distances imply a bigger segmentation error, since the distance between boundaries is measured.

7.3.3 MRI data

Uterine fibroids

Eighteen patients affected by symptomatic uterine fibroids who underwent Magnetic Resonance guided Focused Ultrasound Surger (MRgFUS) therapy [362] were considered. The total number of the examined fibroids was 29, overall represented on 163 MR slices, since some patients presented a pathological scenario with multiple fibroids. The analyzed images were acquired using a Signa HDxt 1.5 T MRI scanner (General Electric Medical Systems, Milwaukee, WI, USA) at two different institutions. These MRI series were acquired after the MRgFUS treatment, executed with the ExAblate 2100 (Insightec Ltd., Carmel, Israel) HIFU equipment. The considered MR slices were scanned using the T1-weighted (T1w) "Fast Spoiled Gradient Echo + Fat Suppression + Contrast mean" (FSPGR+FS+C) sequence. This MRI protocol is usually employed for Non-Perfused Volume (NPV) assessment, since ablated fibroids appear as hypo-intense areas due to low perfusion of the contrast mean [291]. Sagittal MRI sections were processed, in compliance with the current clinical routine for therapy response assessment [291]. In current clinical practice, the NPV evaluation procedure is fully manual [375]. Two uterine fibroid MR slices are depicted in Figures 7.1a and 7.1b.

Brain metastatic tumors

Twenty-seven brain metastases treated using a Leksell Gamma Knife (Elekta, Stockholm, Sweden) stereotactic neuro-radiosurgical device [251] were processed, for a total of 248 MR slices. All the available MRI datasets were acquired on a Gyroscan Intera 1.5 T MR Scanner (Philips Medical System, Eindhoven, the Netherlands), before treatment, for the planning phase. In current radiation therapy practice, Gamma Knife treatments are planned manually by a neurosurgeon on MRI alone, by typically using T1w Fast Field Echo (T1w FFE) CE-MRI sequences [372, 376]. Thanks to the Gadolinium-based contrast agent, brain lesions appear as enhanced hyper-intense zones. Sometimes a dark area might be present due to either edema or necrotic tissues [372, 374]. Two representative instances of brain tumors are shown in Figures 7.1c and 7.1d.

7.4 Results

This section presents the experimental results achieved by MedGA. We first analyzed the performance of MedGA by varying the parameter settings of the GA underlying our methodology; we then compared it against the most common and popular image enhancement techniques in the image processing field (see [165] for additional details). Finally, we describe the integration of MedGA as pre-processing step into a segmentation pipeline and the achieved results.

The performance of MedGA were compared against the following image enhancement techniques:

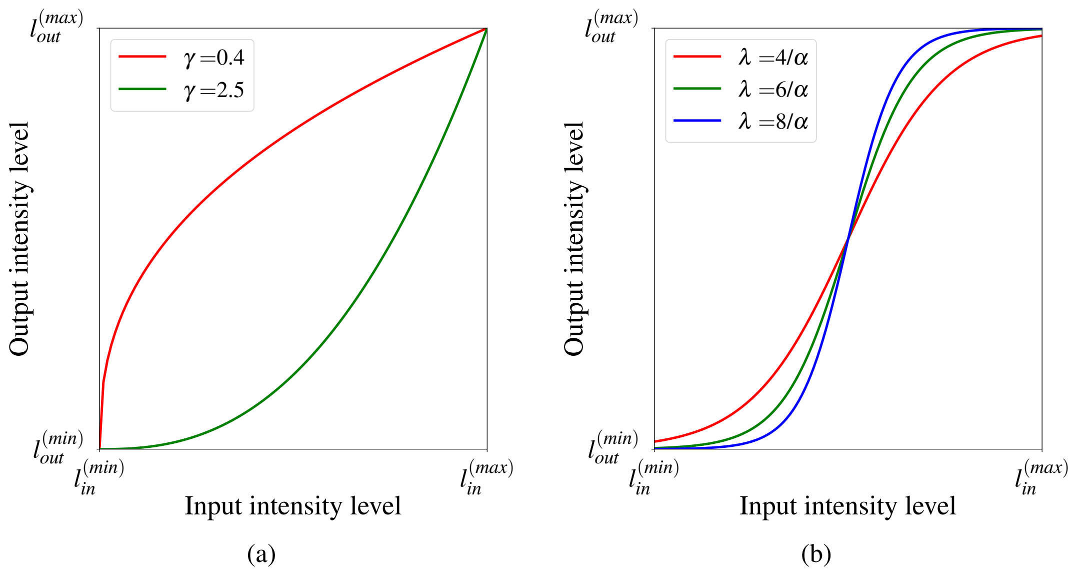


Fig. 7.6 Plots of the implemented global non-linear intensity transformations for image enhancement: (a) Gamma Transformation; (b) Sigmoid intensity Transformation. We report on the x-axis the input intensity range $[l_{in}^{(min)}, \dots, l_{in}^{(max)}]$, and on the y-axis the output intensity range $[l_{out}^{(min)}, \dots, l_{out}^{(max)}]$.

- Histogram Equalization (HE) [331, 180], which adjusts pixel intensities for contrast enhancement according to the normalized histogram of the original image I_{orig}. With HE, gray levels are more uniformly distributed on the histogram, by spreading the most frequent intensity values;
- Bi-Histogram Equalization (Bi-HE) [227]—a modification of the traditional HE—that addresses issues concerning mean brightness preservation;
- Gamma Transformation (GT), which is a non-linear operation using the power-law relationship $s(r) = cr^{\gamma}$, where r and s are the input and the output gray-scale values, respectively, and c is a multiplication constant ($c = 1$ in the following tests). The parameter γ is set to values greater than 1 (i.e., decoding gamma) to obtain a gamma expansion, or to values smaller than 1 (i.e., encoding gamma) to realize a gamma compression (see Figure 7.6a). In our tests we considered the values $\gamma = 0.4$ and $\gamma = 2.5$, as higher (lower) values of γ tend to logarithmic (anti-logarithmic) functions, resulting in an excessively bright (dark) output image, unsuitable for practical medical applications [144];

- Sigmoid intensity Transformation (ST) function (Figure 7.6b), also called S-shaped curve, which is a global non-linear mapping defined as follows:

$$s(r) = \frac{l_{in}^{(max)}}{1 + \exp\left(-\lambda(r - \alpha)\right)}, \tag{7.16}$$

  where $l_{in}^{(max)} = \max\{\mathcal{L}_{in}\} = \max\{\mathcal{L}'_{in}\}$ is the asymptotic maximum value of the function, $\alpha = \frac{1}{2}\left(l_{in}^{(max)} - l_{in}^{(min)}\right)$ is the midpoint value, and λ defines the function steepness. This transformation stretches the intensity around the level α, by making the hypo-intense histogram part darker and the hyper-intense histogram part brighter. Thus, the difference between the minimum and maximum gray values and the gradient magnitude of the image are increased, obtaining strong edges [144]. In our tests, we used sigmoid functions that allow for considering the entire input dynamic range, by varying the curve slope with the values $\lambda \in \left\{\frac{4}{\alpha}, \frac{6}{\alpha}, \frac{8}{\alpha}\right\}$.

7.4.1 GA setting analysis

To analyze the performance of MedGA and identify the best settings for the image enhancement problem, we considered a calibration set consisting of 80 medical images randomly selected from the available fibroid MRI data, and we varied the settings of MedGA used throughout the optimization process, that is: (*i*) the size of the population $|P| \in \{50, 100, 150, 200\}$; (*ii*) the crossover probability $p_c \in \{0.8, 0.85, 0.9, 0.95, 1.0\}$; (*iii*) the mutation probability $p_m \in \{0.01, 0.05, 0.1, 0.2\}$; (*iv*) the size of the tournament selection strategy $k \in \{5, 10, 15, 20\}$. In all tests MedGA was run for $T = 100$ iterations. Each MedGA execution was performed by varying one setting at a time, for a total of 320 different settings tested and a total number of $320 \times 80 = 25600$ MedGA executions.

The results of these tests highlighted that, for each value of $|P|$, the best settings in terms of fitness values achieved are:

1. $|P| = 50$, $p_c = 0.85$, $p_m = 0.01$, $k = 15$;
2. $|P| = 100$, $p_c = 0.9$, $p_m = 0.01$, $k = 20$;
3. $|P| = 150$, $p_c = 0.85$, $p_m = 0.01$, $k = 20$;
4. $|P| = 200$, $p_c = 0.85$, $p_m = 0.01$, $k = 20$.

Figure 7.7 reports the comparison of the performance achieved by MedGA with these settings, where the Average Best Fitness (ABF) was computed by taking into account, at each iteration

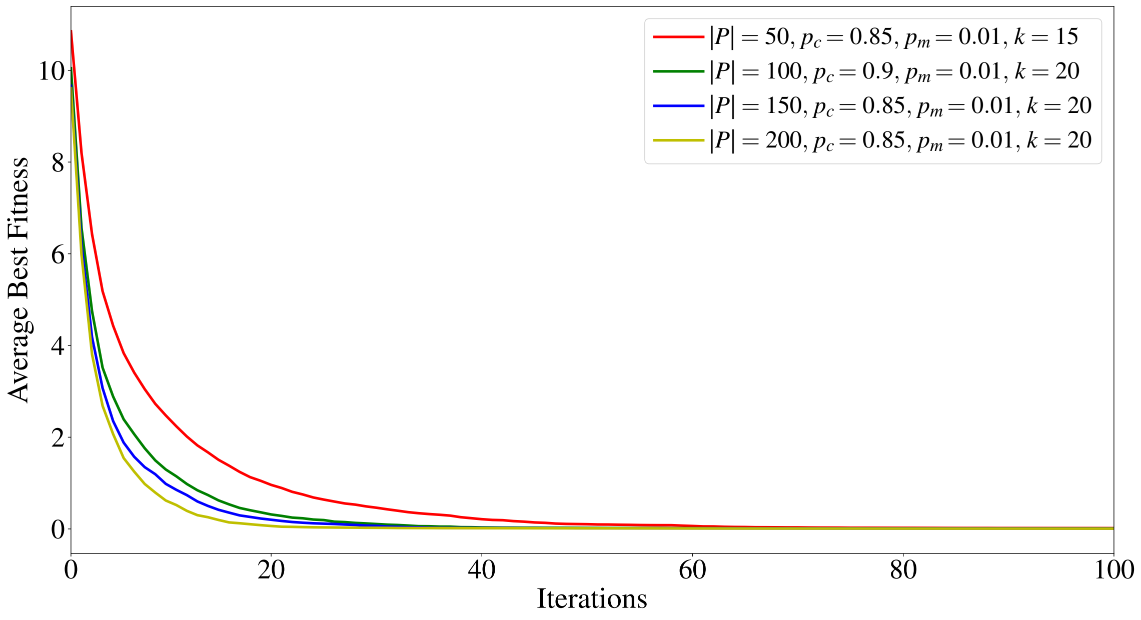


Fig. 7.7 Comparison of the ABF achieved by MedGA with the best parameterizations found for each value of $|P|$ tested here. The average was computed over the results of the optimization of 80 MR images.

of MedGA, the fitness value of the best individuals over the 80 optimization processes. It is clear from the plot that, despite the final ABF values are comparable in all settings, the convergence speed increases with the size of the population, as well as the running time required by MedGA; therefore, to choose the best settings, we analyzed the computational performance concerning the 4 tests listed above.

Considering that an MRI series related to a single patient contains on average 10 slices with ROI fibroids, we tested the clinical feasibility of MedGA by calculating the total execution time for enhancing 10 randomly chosen MR images. For what concerns the tests 1-4 described above, the executions lasted on average 672.12 s, 1290.15 s, 1987.8 s, and 2669.74 s, respectively, for the optimization of the same batch of 10 images running on a single-core of the Intel® Xeon® E5-2440 CPU with 2.40 GHz clock frequency (16 GB RAM and CentOS 7 operating system) of one node of the Advanced Computing Center for Research and Education (ACCRE) at Vanderbilt University, Nashville, TN, USA. On the other hand, by exploiting the 6 cores of the same CPU to execute the parallel version of MedGA, we achieved up to 3.6× spped up with respect to the sequential version. The results achieved using the parallel version of MedGA confirm the importance of HPC solutions in the field of real healthcare environment to obtain clinically feasible outcomes, that is, enhancing MR images in reasonable time for medical imaging practice.

Taking into account both the performance of MedGA in terms of ABF and the running time required to process 80 images, we selected the parameter settings $|P| = 100$, $p_c = 0.9$, $p_m = 0.01$, $k = 20$ as the best trade-off characterized by a good convergence speed and an adequate running time (for this specific application), and we exploited this configuration for all tests reported and discussed in the following sections.

7.4.2 Enhancement results

In order to achieve a comprehensive comparison between MedGA and the other pre-processing techniques listed above, we exploited the entire set of MRI data consisting in 18 patients affected by uterine fibroids and 27 brain metastatic cancers.

Tables 7.1 and 7.2 show the enhancement/pre-processing results achieved by each method on the uterine fibroid and brain tumor MRI datasets, respectively, by using the metrics reported in Section 7.3.1.

Considering the results in Table 7.1, HE over-enhances the processed uterine fibroid MR images, as denoted by the highest mean *#DE* value, while Bi-HE allows for the preservation of the mean brightness, as also indicated by the lower mean value of *AMBE*. For what concerns the other techniques, on the one hand, GT with $\gamma = 0.4$ yields better results compared to GT with $\gamma = 2.5$, especially in the case of the *SSIM*; on the other hand, all metrics related to the tested ST functions show that their performances decrease as the value of λ increases. This phenomenon is related to the rapid variation characterizing the highest values of λ, which do not allow for taking into consideration the existing dependency among the pixels, especially those in the neighborhood. As it can be observed in Figures 7.10 and 7.11, MedGA strengthens the ROI edges by enhancing details and features useful for image binarization; this result confirms, from a qualitative perspective, the quantitative results presented above. From an overall view of the metrics values, we can claim that the approaches obtaining the highest values of the *#DE* measure (i.e., HE and GT with $\gamma = 2.5$) could imply a considerable over-enhancement of the output image, according to the other image quality metrics.

The results on brain tumor MRI data reported in Table 7.2 show a slightly different trend, also due to the small size of the pre-processed cropped sub-images. As a first evidence, both GTs do not preserve the input mean brightness considering the *AMBE* measure. Interestingly, GT with $\gamma = 2.5$ and $\gamma = 0.4$ achieve the highest and the lowest *#DE* values, respectively. Bi-HE strongly improves the enhancement metrics obtained by HE, by generally reporting the best results. Consistently with the metrics calculated on uterine fibroid MRI data, all the results concerning ST functions get worse when the value of λ increases. The highest *SSIM* mean value is achieved by Bi-HE, revealing the best structural information, even though MedGA obtains the best signal quality in terms of *PSNR* mean values.

Table 7.1 Values of the image enhancement evaluation metrics achieved by MedGA and the other implemented approaches on the uterine fibroid MRI series. The experimental results are calculated on the MRI dataset with 18 patients affected by uterine fibroids and expressed as mean and standard deviation values. Numbers in bold indicate the best value for each measure achieved by the implemented methods.

Method	PSNR		#DE		AMBE		SSIM	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
HE	30.994	1.949	**975.465**	475.951	0.085	0.029	0.859	0.044
Bi-HE	31.880	2.046	907.177	415.703	0.038	0.020	0.907	0.032
GT $\gamma = 0.4$	30.194	2.170	717.555	401.186	0.212	0.019	0.823	0.024
GT $\gamma = 2.5$	29.952	2.127	965.012	380.967	0.261	0.012	0.586	0.075
ST $\lambda = 4/\alpha$	33.971	1.874	872.594	396.016	0.040	0.014	0.880	0.023
ST $\lambda = 6/\alpha$	32.286	1.975	869.032	378.277	0.060	0.021	0.715	0.056
ST $\lambda = 8/\alpha$	31.353	2.029	841.420	348.674	0.073	0.025	0.613	0.070
MedGA	**37.366**	2.347	866.604	409.604	**0.033**	0.011	**0.928**	0.025

Table 7.2 Values of the image enhancement evaluation metrics achieved by MedGA and the other implemented approaches on the brain tumor MRI series. The experimental results are calculated on the MRI dataset composed of 27 brain tumors and expressed as mean and standard deviation values. Numbers in bold indicate the best value for each measure achieved by the implemented methods.

Method	PSNR		#DE		AMBE		SSIM	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
HE	34.215	1.447	38.779	21.287	0.124	0.049	0.756	0.112
Bi-HE	36.758	1.718	44.923	26.252	**0.042**	0.024	**0.932**	0.021
GT $\gamma = 0.4$	33.193	0.841	21.271	18.467	0.229	0.020	0.713	0.065
GT $\gamma = 2.5$	33.520	1.113	**45.119**	27.444	0.229	0.028	0.457	0.096
ST $\lambda = 4/\alpha$	36.812	0.923	43.574	27.307	0.055	0.019	0.848	0.048
ST $\lambda = 6/\alpha$	35.270	0.942	43.779	26.492	0.079	0.028	0.645	0.105
ST $\lambda = 8/\alpha$	34.435	0.991	44.072	26.508	0.090	0.034	0.543	0.121
MedGA	**37.751**	1.990	43.534	22.598	0.079	0.039	0.881	0.053

These findings are also corroborated by a visual inspection of Figures 7.10 and 7.11, where the enhanced images using HE and GT with $\gamma = 2.5$ present an inadequate appearance for image observation and interpretation. Overall, these results highlight that MedGA significantly outperforms the conventional image enhancement approaches in terms of signal quality and perceived structural information in the images, while preserving the input mean brightness. As a matter of fact, in both MR image analysis applications, MedGA remarkably achieves the highest *PSNR* mean values with respect to the state-of-the-art methods, generally involving the highest signal quality. Concerning the other enhancement

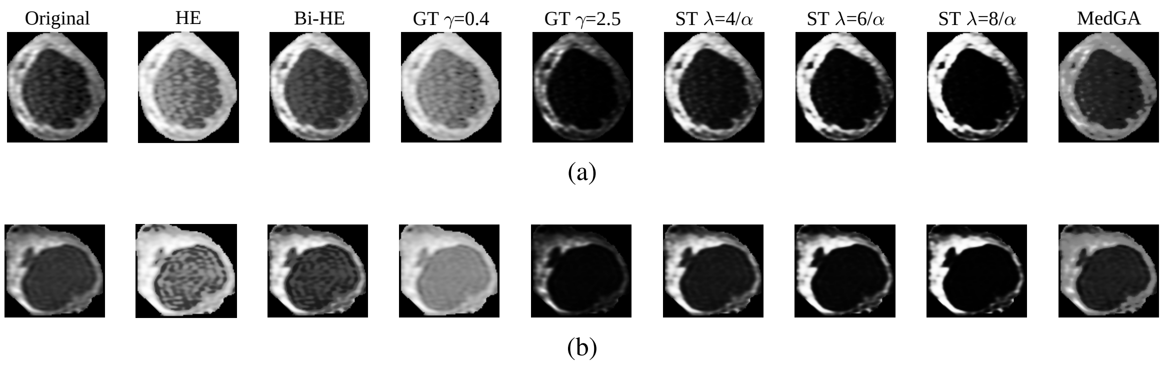


Fig. 7.8 Examples of enhancement achieved on medical images of uterine fibroids by HE, Bi-HE, GT (with $\gamma \in \{0.4, 2.5\}$), ST (with $\lambda \in \{4/\alpha, 6/\alpha, 8/\alpha\}$) and MedGA, compared with the original input MR image.

techniques, we observe that Bi-HE achieves better performance with respect to HE, while increasing the values of γ and λ corresponds to a degradation of the performances of GT and ST, respectively.

7.4.3 Segmentation pipeline

Image enhancement techniques can facilitate the user interpretation of an image as well as improve the automated image understanding. Considering that achieving an effective MR image segmentation is an important processing goal [411], the enhanced images after applying MedGA are here segmented using the IOTS algorithm [359, 445], which is the simplest automated segmentation approach. To this aim, we developed two slightly different post-processing pipelines to refine the results achieved by this efficient adaptive thresholding technique (see Figure 7.9). These post-processing steps, here applied to perform uterine fibroid and brain tumor segmentation are described in what follows. MedGA is able to enhance images in segmentation tasks involving both hyper- and hypo-intense ROIs in CE-MR images, also dealing with data unbalanceness (i.e., the number of foreground pixels is either much higher or lower than the number of background pixels).

Uterine fibroid segmentation

Firstly, uterus region delineation is required. This task can be accomplished manually by the user or automatically by means of computational methods to reduce operator-dependency, as described in [292]. ROIs are represented by tissues with low contrast mean absorption (i.e., NPV), thus the pixels with lower values with respect to the achieved threshold are yielded in

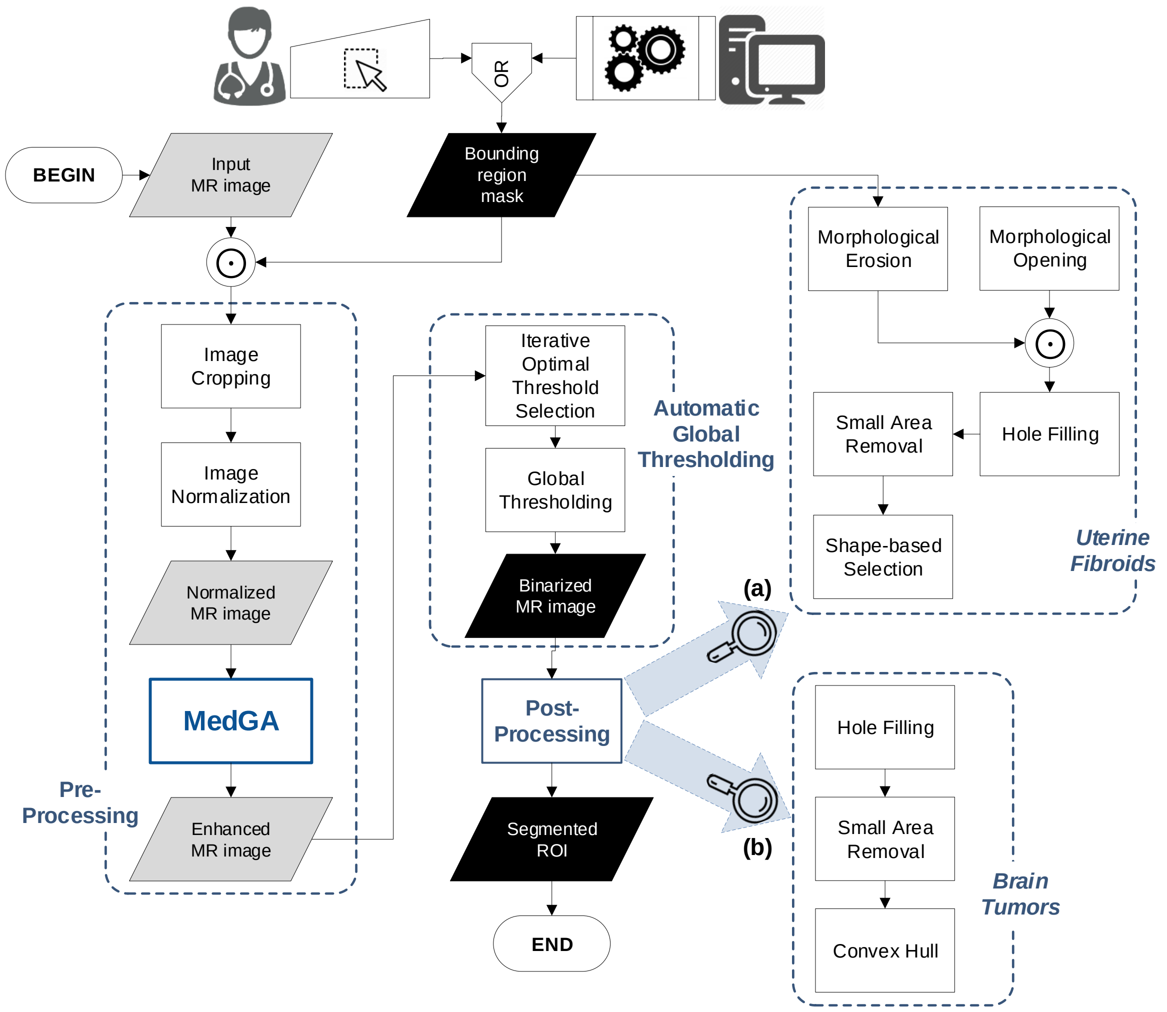


Fig. 7.9 Flow diagram of the proposed pipeline that integrates MedGA as a pre-processing step for MR image segmentation based on the efficient IOTS algorithm [359, 445]. Note that two slightly different post-processing pipelines were developed for (a) uterine fibroids and (b) brain tumors. Gray and black data blocks denote MR gray-scale images and binary masks, respectively.

the binarized MR image. However, segmentation approaches have to take into account NPV inhomogeneities, due to sonication spots during the MRgFUS treatment.

The used post-processing refinement steps are the following (Figure 7.9a):

1. morphological opening with a circular structuring element (2-pixel radius) to separate possible loosely connected hypo-intense regions;

2. some regions at the boundary of the uterus bounding region mask could present similar intensity values to gray levels characterizing fibroid regions, so being included in the thresholding output. To eliminate this ambiguity, it is appropriate to apply a morphological erosion (with a circular structuring element of 5-pixel radius) to the ROI binary mask, and then the logical pixel-by-pixel product (i.e., Hadamard multiplication) with the image resulting from the previous step is performed;

3. a hole filling algorithm is necessary to deal with possible holes in fibroid regions also due to non-uniform distribution of ablated tissue caused by sonication spots;

4. segmentation is further improved through a connected-component labeling based operation by removing objects that are smaller than a certain area (i.e., 120 pixels) and characterized by similar intensity with respect to the fibroids to be treated, because there may be regions or artifacts caused by very small dark areas;

5. some lengthened connected-component with sufficiently large area could be present (i.e., due to other anatomical structures or to acquisition artifacts). Fibroids, in fact, present a spherical or semi-spherical shape [452] that can be denoted by means of the parameters of the various connected-components. This connected-component based selection considers the eccentricity (that is, the ratio between the foci distance related to an ellipse and its own major axis length) and the extent (that is, the ratio between the pixels belonging to the region and the bounding box pixels) of the detected regions. Specifically, experimental reference values to discern fibroids from the rest of connected-components are: $0.3 \leq \text{extent} < 0.8$ and $0.0 \leq \text{eccentricity} < 0.8$, according to [292]. Lastly, any connected-component, which has passed the shape-based control and whose centroid distance is more than a given upper limit (i.e., $\frac{\sqrt{M^2+N^2}}{3}$) from the MR image center, is removed.

Brain tumor segmentation

The accurate and reproducible measurement of tumor size and its changes over time is crucial for diagnosis, treatment planning, as well as monitoring of response to oncologic therapy for brain tumors [283]. As a preliminary step, the user has to interactively select a bounding region that includes the tumor zone (by means of a free-hand “lasso” tool). Since the areas to segment are enhancement regions, the pixels that have higher intensities than the threshold are selected during the image binarization phase. Brain metastatic cancers may contain also necrotic cores, which could affect the achieved enhancement region segmentation. Therefore, some refinement steps are useful to cope with this situation.

The used post-processing pipeline is described in the following (see Figure 7.9b):

1. hole filling algorithm to consider also necrotic areas;

2. adaptive post-processing steps based on the size of the input image, consisting in small area removal (considering 4-connectivity) with minimum threshold equal to 30 pixels on images with size greater than 300 pixels, or 10 pixels otherwise;

3. to allow also for large bounding regions, shape-based selection is applied in the case of at least two connected-components, according to: extent ≥ 0.6 and $0.0 \leq$ eccentricity $<$ 0.8 (see [376]). However, when a single connected-component is present, these controls are avoided;

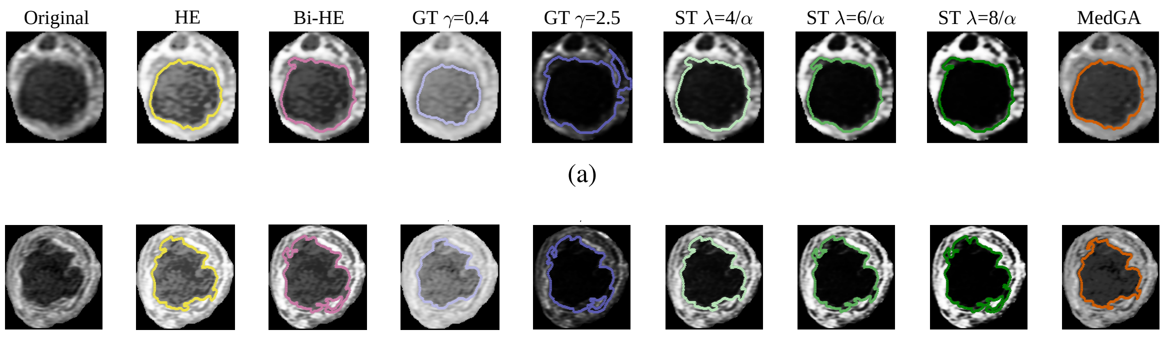
4. brain metastases have a pseudo-spherical appearance [14], therefore a convex hull algorithm is employed to envelope the segmented lesion into the smallest convex polygon containing this region.

7.4.4 Medical image segmentation results

Figures 7.10 and 7.11 show two examples of uterine fibroid and brain tumor MR images, respectively, which were pre-processed by means of the comparison methods and segmented using the processing pipelines described above.

The quantitative segmentation results achieved by using the pipeline in Figure 7.9a, employing the different pre-processing approaches, on the analyzed MRI dataset composed of 18 patients affected by uterine fibroids are depicted in the boxplots in Figure 7.12, reporting both overlap-based and distance-based metrics values. Analogously, the boxplots concerning the segmentation results, achieved by using the pipeline in Figure 7.9b on the analyzed MRI dataset consisting in 27 brain metastases, are shown in Figure 7.13.

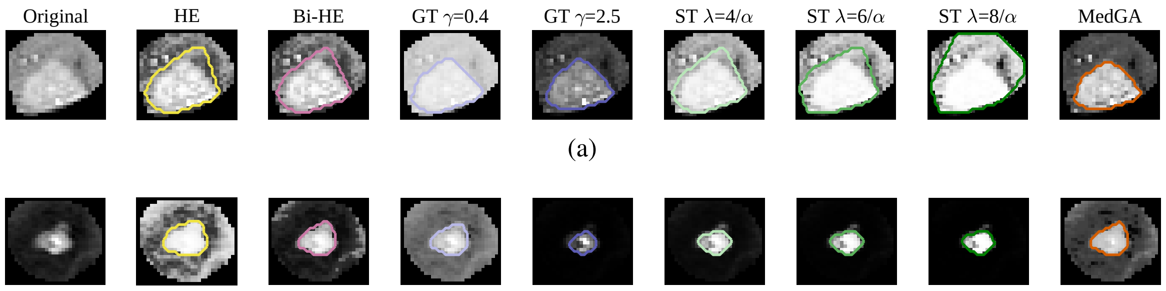
In the literature, it has been shown that a *DSI* above 70% is generally regarded as a satisfactory level of agreement between two segmentations (i.e., manual and automated delineations) in clinical applications [492]. Since the MR image segmentation methods obtain a *DSI* appreciably higher than 70% regardless the pre-processing technique, we can consider that the processing pipelines in Figures 7.9 are clinically valuable, thus allowing for



(a)

(b)

Fig. 7.10 Segmentation results achieved on the uterine fibroids in Figures 7.1a and 7.1b by the processing pipeline in Figure 7.9a exploiting the implemented image enhancement approaches (namely: HE, Bi-HE, GT $\gamma = 2.5$, GT $\gamma = 0.4$, ST $\lambda = 4/\alpha$, ST $\lambda = 8/\alpha$ and ST $\lambda = 6/\alpha$) and the MedGA.



(a)

(b)

Fig. 7.11 Segmentation results achieved on the brain tumors in Figures 7.1c and 7.1d by the processing pipeline in Figure 7.9b by exploiting the implemented image pre-processing approaches (namely: HE, Bi-HE, GT $\gamma = 2.5$, GT $\gamma = 0.4$, ST $\lambda = 4/\alpha$, ST $\lambda = 8/\alpha$ and ST $\lambda = 6/\alpha$) and the MedGA.

a fair comparison on segmentation performance among the state-of-the-art pre-processing algorithms.

In both cases, the segmentation results concerning the images pre-processed using MedGA achieved the highest mean and median *DSI* values, with low standard deviation. GT with $\gamma = 0.4$ and Bi-HE obtained the second best performances for uterine fibroid and brain tumor MR image segmentation, respectively. So, we can claim that MedGA shows the highest accuracy and reliability in the two considered MRI analysis tasks. This evidence is also confirmed by the boxplots, where the distributions for MedGA present significantly less than 10% outliers in all the overlap-based metrics, thus evidencing extremely low statistical dispersion. As a matter of fact, MedGA is the only technique that significantly supports the IOTS algorithm in dark (i.e., uterine fibroid NPV) and bright (i.e., brain tumor enhancement

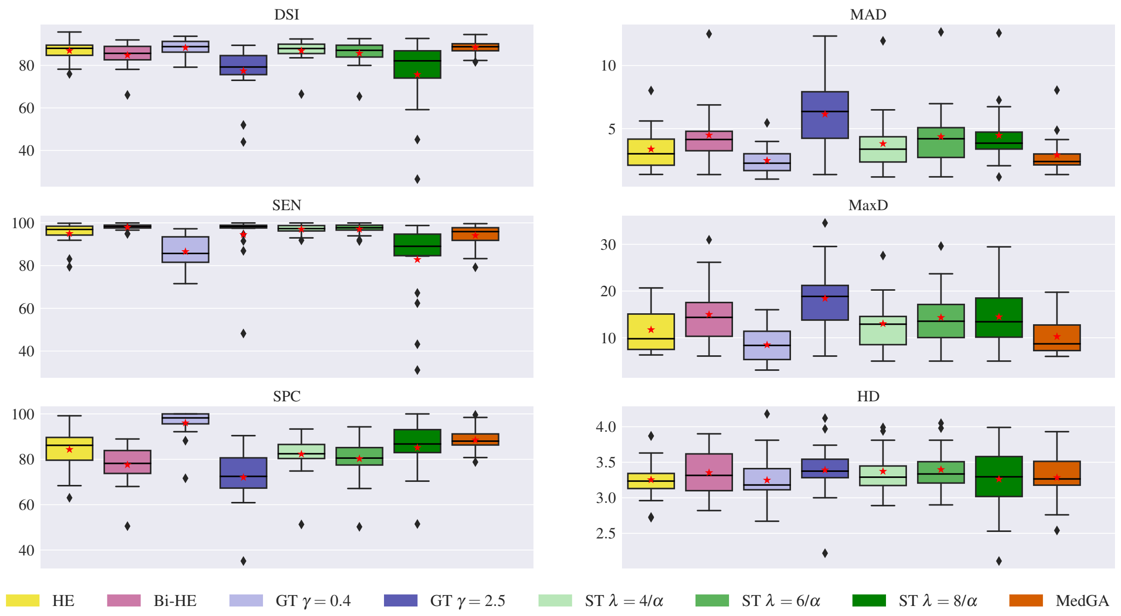


Fig. 7.12 Boxplots of overlap-based and distance-based metrics (left and right columns, respectively) obtained on the MRI dataset composed of 18 patients with uterine fibroids who underwent MRgFUS treatment. The lower and the upper bounds of each box represent the first and third quartiles of the statistical distribution, respectively. The median (i.e., the second quartile) and the mean values are represented by a black solid line and a red star, respectively. Whisker value is 1.5 in all cases and outliers are displayed as black diamonds.

region) ROI extraction. In agreement with the image enhancement results discussed in Section 7.4.2, GT with $\gamma = 0.4$ considerably outperforms GT with $\gamma = 2.5$. The decreasing trend, related to ST when the value of λ increases, is also confirmed. ST with $\lambda = 4/\alpha$ achieved good results in both cases. Brain tumor MR images pre-processed by means of HE achieved low *DSI* values, but better results are obtained on uterine fibroid MR segmentation with respect to Bi-HE. Overall, the achieved segmentation performance shows the great accuracy and reliability of the proposed EC-based computational model. Considering *SEN* and *SPC*, MedGA yielded the best trade-off between these two often conflicting measures that should be always considered and combined together. These metrics reveal that the other techniques could involve over- or under-segmentation.

The achieved spatial distance-based indices are consistent with overlap-based metrics, also observing the corresponding boxplots shown in Figures 7.12 and 7.13. Hence, MedGA allows also for accurate results in terms of distance between the automated and the manual boundaries. It is worth noting that, generally, the boxplots pertaining to MedGA results present the lowest statistical dispersion (in terms of box width and number of outliers), which implies a lower standard deviation with respect to the conventional techniques. Therefore,

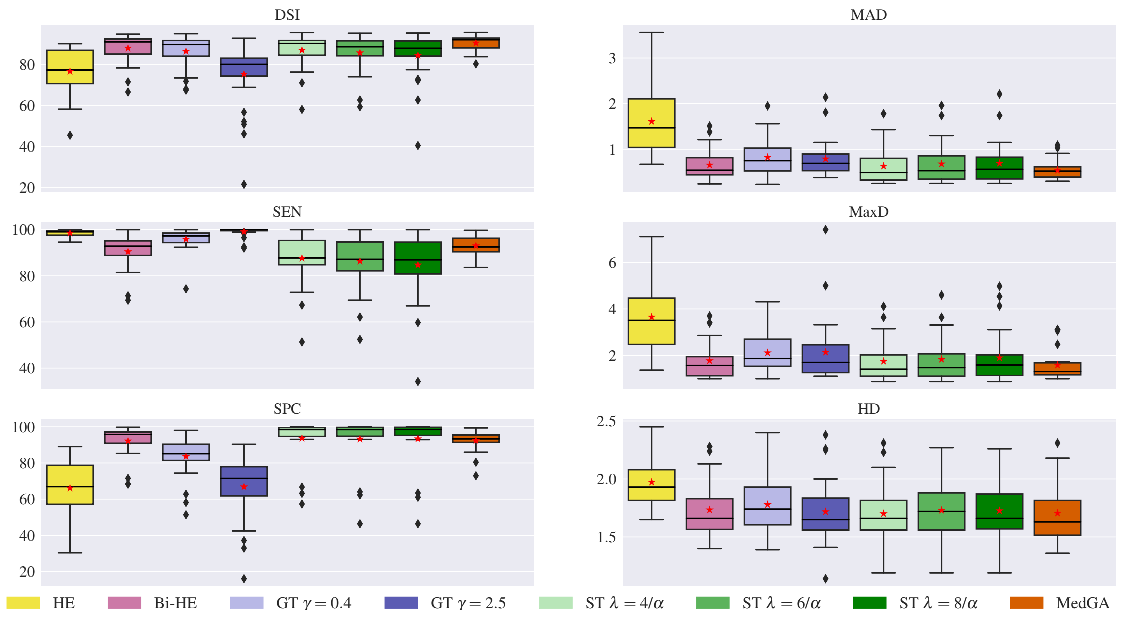


Fig. 7.13 Boxplots of overlap-based and distance-based metrics (left and right columns, respectively) obtained on the MRI dataset composed of 27 brain metastatic tumors underwent stereotactic neuro-radiosurgery. The lower and the upper bounds of each box represent the first and third quartiles of the statistical distribution, respectively. The median (i.e., the second quartile) and the mean values are represented by a black solid line and a red star, respectively. Whisker value is 1.5 in all cases and outliers are displayed as black diamonds.

the use of MedGA as a pre-processing step allows for considerably robust and reliable segmentation results. These experimental findings are endorsed by the segmentation examples shown in Figures 7.10 and 7.11.

7.5 Conclusions

In order to overcome the limitations related to the assumptions underlying threshold selection methods and automatically determine a suitable optimal threshold, MedGA tackles this complex problem, introducing a fitness function tailored to better separate the two underlying sub-distributions of the gray level intensities. Unlike the traditional image enhancement techniques that generally improve the contrast level of the whole image, MedGA focuses on MR image sub-regions characterized by a roughly bimodal histogram, making it valuable in clinical contexts, especially involving CE-MRI analysis. As a matter of fact, the Computational Intelligence approaches presented in [117, 188] explicitly consider in the fitness function both the number of edge pixels and the intensity of these pixels, thus achieving high

#DE values that would consistently lead to over-enhanced images, possibly yielding also inaccurate ROI segmentations.

We integrated MedGA as a pre-processing stage in a pipeline for image enhancement, automatic global thresholding and segmentation, and we tested it on two different clinical scenarios requiring CE-MR image analysis: (*i*) uterine fibroid segmentation in MRgFUS treatments, and (*ii*) brain metastatic cancer segmentation in neuro-radiosurgery therapy. Overall, the MedGA pre-processing outperformed, in terms of image quality and segmentation accuracy, the conventional image enhancement, namely, HE, Bi-HE, GT, and ST. According to the achieved experimental results, MedGA was shown to be an appropriate and reliable solution when employed as a medical image pre-processing method. Even considering the statistical dispersion of the segmentation evaluation metrics, MedGA achieved the most robust and repeatable segmentation results.

Considering the achieved results in terms of #DE, the performance of MedGA could be further improved—in terms of contrast—by integrating a novel component in the fitness function, which explicitly relies on the number of detected edges. Since this additional component would have a different purpose and a different magnitude, a multi-objective optimization method should be taken into account. In particular, MedGA could be extended by means of an effective Evolutionary Computation approach, such as NSGA-III [108], to simultaneously optimize both conflicting objectives, which consist in maximizing the number of edges while minimizing the distance between the optimal threshold and the two normal distributions.

Conclusions

Objectives of the work

The work discussed in this thesis has been motivated by the need to tackle complex problems in Life Sciences—especially in the biomedical field—which require novel computational methods capable of dealing with both the lack of quantitative information and the availability of huge amounts of data. Indeed, we presented different approaches based on Evolutionary Computation (EC) and Swarm Intelligence (SI) along with High-Performance Computing (HPC) solutions to address Systems Biology, Bioinformatics, and Medical Imaging challenges. In particular, we proposed:

1. two deterministic simulators accelerated on Graphics Processing Units (GPUs) and a stochastic simulator exploiting the Many Integrated Cores (MIC) coprocessors to analyze the emergent dynamics of biological systems;

2. different methodologies based on SI to solve the Parameter Estimation (PE) problem considering both single and multiple targets;

3. a novel computational method relying on Genetic Algorithms (GAs), accelerated by means of a Master-Slave approach, to solve the Haplotype Assembly problem;

4. a novel approach based on GAs for the enhancement and segmentation of Magnetic Resonance (MR) images, which are characterized by a bimodal gray level intensity histogram.

Contribution to Systems Biology

The proposed simulators are valuable tools for in-depth investigations of biochemical systems, since they strongly reduce the running time required by the computational analyses performed in Systems Biology, which generally rely on a massive number of simulations (e.g., Parameter Sweep Analysis [307] and Sensitivity Analysis [70]). This computational demand can become prohibitive for the analysis of large-scale biological systems, characterized by hundreds or thousands of reactions and molecular species.

We first proposed LASSIE (LArge-Scale SImulator), a GPU-powered simulator of large-scale biochemical systems based on mass-action kinetics. LASSIE is a "black-box" simulator able to automatically convert Reaction-Based Models (RBMs) of biological systems into the corresponding system of Ordinary Differential Equations (ODEs). In particular, to solve systems of ODEs characterized by stiffness, LASSIE automatically switches between the Runge–Kutta–Fehlberg (RKF) and the Backward Differentiation Formulae (BDF) integration methods. LASSIE adopts a novel fine-grained parallelization strategy to distribute on the GPU cores all the calculations required to solve the system of ODEs. By virtue of this implementation, LASSIE achieves up to 92× speed-up with respect to the Livermore Solver of Ordinary Differential Equations (LSODA) ODE solver [341], therefore reducing the running time from approximately 1 month down to 8 hours to simulate models consisting in, for instance, four thousands of reactions and species. Notably, thanks to its smaller memory footprint, LASSIE is able to perform fast simulations of even larger models, whereby the tested CPU-implementation of LSODA failed to reach termination.

Even if LASSIE represents an important achievement in this research field, it performs in parallel the calculations of a single simulation. To overcome this limitation, we designed and developed FiCoS (Fine- and Coarse-grained Simulator), a GPU-powered simulator that exploits both a fine- and a coarse-grained parallelization strategy. With FiCoS, the system of ODEs is solved by using two Runge-Kutta methods of order 5: the DOPRI5 method in the absence of stiffness, and the RADAU5 method when the system is stiff. FiCoS can distribute a massive number of simulations, as well as all the calculations required by each simulation, over the available GPU cores. This mixed parallelization strategy takes advantage of the dynamic parallelism provided by the latest Nvidia GPUs and leverages their massive parallel capabilities, which are essential to obtain a relevant reduction of the computational costs. Considering the different tests executed to assess the performance of FiCoS, we observed that it resulted 487× faster in terms of simulation time and 855× faster considering the integration time with respect to Variable-coefficient ODE solver (VODE) [53]. The simulation time required by LSODA resulted 366× higher, while the integration time 79×, compared to those obtained by FiCoS, which performed 760× faster than LASSIE in terms of integration time and 298× considering the simulation time. Finally, FiCoS was also capable of outperforming cupSODA [306], obtaining a speed-up up to 17× considering the integration time and up to 7× in terms of simulation time.

In order to fully exploit the Compute Unified Device Architecture (CUDA), the memory hierarchy must be exploited as much as possible. Because of the peculiar sequential structure of both explicit and implicit integration algorithms, LASSIE and FiCoS kernels are lightweight and rarely reuse any variables. For this reason, the current implementations only leverage the global memory (characterized by high latencies) and registers to manipulate the mutable data. The shared memory has not been exploited in any way, leaving room for potential future improvements of performance. In a future release, we plan to make use of these memories, for instance to store the array of the kinetic constants and the structures containing the offsets used to correctly decode the ODEs. Both LASSIE and FiCoS currently exploit only a single GPU, even on multi-GPU systems; as a future improvement of these works, we plan to extend their implementation in order to support multi-GPU systems, to further increase the size of the models that can be simulated.

In this thesis we also presented a simulator implementing the Stochastic Simulation Algorithm (SSA), accelerated by means of MIC coprocessors, which exploit a coarse-grained parallelization. We performed several tests and compared MIC with two alternative architectures: CPUs and GPUs. According to our results, MICs provide better performance than CPUs when more than 10 SSA runs are executed, although GPUs outperform both MICs and CPUs when more than 80 SSA runs are executed. It is worth noting that the running time on GPUs remains constant until computing resources (i.e., the cores) are available; in the case of MICs, the performance scale well until the maximum number of supported threads (i.e., 240) is reached. Moreover, the performance of MICs in native mode are strongly improved when vector instructions are exploited, especially in the case of larger synthetic stochastic models. We also tested the *offload* capability of the Xeon Phi coprocessors, that is, the possibility of automatically distributing independent calculations over multiple threads. Our results showed that the offload of the SSA algorithm does not provide a speed-up since the portions of code that can be offloaded consist in a few instructions only; therefore, the offload parallelization is affected by overheads and data transfers. The empirical analyses that we proposed might facilitate the selection of the proper parallel architecture and SSA implementation when a large number of mutually independent simulations are needed, as is the case of many computationally expensive tasks typically carried out for in-depth investigations of biological systems (see, e.g., [11, 44]).

Afterwards, we focused on the role that the kinetic parameters play in driving the dynamic behavior of biochemical systems. As a matter of fact, a small change of their values can dramatically lead to completely different outcomes. In order to calibrate the kinetic parameters characterizing the biochemical systems, we addressed the PE problem by exploiting different EC and SI approaches. In particular, we presented an in-depth analysis of the EC and SI methods, comparing the optimization performance of these methods in solving both classic benchmark functions and the PE problem. In addition, since all the methodologies employed in this thesis to solve the PE problem are population-based and require a massive number of simulations for the fitness evaluations, they were coupled with the GPU-powered biochemical simulators. The achieved results point out that the performance of the meta-heuristics can drastically change according to the context of application, highlighting that the fitness landscape associated with the PE problem is completely different with respect to the fitness landscapes of the most known benchmark functions. We plan to further investigate this issue to understand which are the peculiarities that allow for describing and characterizing the fitness landscape related to the PE, in order to propose a novel population-based approach tailored to this important problem.

Fuzzy Self-Tuning Particle Swarm Optimazation (FST-PSO) resulted one of the best strategies to address the PE problem; we coupled it with FiCoS to solve the PE of 78 unknown kinetic parameters of a large-scale model of the human intracellular metabolic pathway in a red blood cell. The achieved results showed that FST-PSO was capable of successfully estimating the missing parameters, allowing for an accurate reproduction of the network behavior and the execution of predictive *in silico* experiments. Thanks to FiCoS, the required running time achieved a $30\times$ reduction with respect to the same methodology exploiting LSODA as ODE solver running on CPU.

Considering the typical scenario of biological laboratory research in which a set of discrete-time measurements are obtained under different experimental conditions, we proposed a multi-swarm version of PSO. In order to deal with the computational burden required by this analysis, we designed an efficient implementation of this methodology, called MS^{2}PSO, which allows for drastically speeding up the computation by exploiting cupSODA and the Master-Slave distributed programming paradigm to offload all the required calculations onto multiple GPUs (when available) and multi-core CPUs. We showed that MS^{2}PSO is capable of estimating a set of kinetic parameters that correctly reproduces the dynamics of the systems in all the considered experimental conditions, reducing at the same time the required running time. As a future extension of this work, the implementation of MS^{2}PSO will be improved to the aim of fully exploiting the cores of the GPU, thus allowing for a parallelization of the multi-swarm PSO also on platforms equipped with a single GPU. Since all these computational methodologies have been provided without a Graphic User Interface (GUI), we plan to develop a user-friendly toolkit that will hopefully represent a unified and comprehensive platform for computational biologists to define, simulate and calibrate mathematical models of biological systems, and execute advanced analyses in a few hours instead of months.

Contribution to Genome Analysis

To tackle the haplotyping problem, which is a hot topic in Computational Biology and Bioinformatics, we proposed GenHap.

GenHap was implemented with a distributed strategy that exploits a Master-Slave computing paradigm in order to speed up the required computations. The solutions yielded by GenHap are accurate, independently of the number, frequency and coverage of the Single Nucleotide Polymorphisms (SNPs) in the input instances. In addition, GenHap does not require any *a priori* hypothesis about the sequencing error distribution in the reads. GenHap was designed for taking into account datasets produced by third-generation sequencing technologies, such as PacBio RS II and Oxford Nanopore MinION, which are characterized by longer reads and higher coverages with respect to the previous generations. We showed that GenHap is remarkably faster than HapCol [342], achieving approximately a $4\times$ speed-up in the case of Roche/454 instances, and up to $20\times$ speed-up in the case of the PacBio RS II dataset. Even though high-throughput DNA sequencing technologies are paving the way for valuable advances in clinical practice, analyzing such an amount of data still represents a challenging task. This applies especially to clinical settings, where accuracy and time constraints are critical [360]. The third-generation sequencing technologies could therefore highly benefit from GenHap, thanks to its capability in solving large combinatorial problems. Along with the Haplotype Assembly issues, novel challenges—e.g., poliploidity, metagenomics, analysis of cancer cell heterogeneity, and Chromosome Conformation Capture (3C) experiments—require sequencing data with a high coverage. An interesting future trend in Genome Analysis is related to its connection with machine learning. As a matter of fact, deep learning has been successfully applied in population genetic inference and learning informative features of data [393]. Combining population genetics inference and Haplotype Assembly can provide insights on patterns regarding the genetic diversity in DNA polymorphism data, especially for rapid adaptation and selection [193]. An additional issue worth of notice is that, although the integration of various types of information (e.g., electronic health records and genome sequences) conveys a wealth of information, it is giving rise to unique challenges in Bioinformatics analysis even in terms of secure genomic information sharing [48]. With reference to secure Genome-Wide Association Study (GWAS) in distributed computing environments, multi-party computation schemes based on conventional cryptographic techniques achieves limited performance in practice [83]. Therefore, HPC could become an enabling factor also in this context.

Contribution to Medical Imaging

A novel image enhancement method based on GAs, specifically tailored for medical images characterized by a bimodal histogram, was proposed. This computational framework, named MedGA, exploits a fitness function that better reveals the two underlying sub-distributions of the gray level intensities, consequently allowing for an improvement in the results achieved by threshold-based algorithms. Considering the possible clinical applications, MedGA was proven to improve the visual perception of a Region Of Interest (ROI) in MRI data. The solution provided by our intelligent image enhancement system can be beneficial to visually assist physicians in interactive decision-making tasks, as well as to improve the final outcome of downstream automated processing pipelines for useful measurements in the clinical practice [367]. MedGA also deals with the practical problems regarding the interpretability of the results yielded by advanced Machine Learning and Computational Intelligence methods in medicine [61]. Indeed, the final best solution found by MedGA (i.e., the output gray level histogram) and the corresponding enhanced image are understandable by physicians. In addition, the efficient encoding of the individuals coupled with effective HPC solutions allows for a clinically feasible computational framework.

In addition, MedGA can be used as an intelligent pre-processing step, in a pipeline defined to realize an efficient threshold-based image segmentation with two classes (i.e., binarization), applied to MRI data. In Pattern Recognition, among the low-level intensity-based techniques, the most straightforward unsupervised image segmentation technique is global thresholding [477]. Image thresholding approaches performed on contrast-enhanced MR image regions could considerably benefit from input data pre-processed by MedGA. As a matter of fact, MedGA tackles the limitations related to the assumptions underlying threshold selection methods, and automatically determine a suitable optimal threshold by exploiting a fitness function tailored at better separating the two underlying sub-distributions of the gray level intensities. MedGA has been currently used only for off-line image analysis. Our Python-coded implementation requires approximately two minutes for each image, using the selected GA parameters. In the case of tomography image stack analysis, we achieved a sublinear speed-up with respect to the number of the available cores by developing a Master-Slave version to distribute on multiple cores the computations pertaining different slices. An additional speed-up could be definitely obtained by porting the current `Python` code into a faster compiled programming language (e.g., `C/C++`) [32]. Consequently, by leveraging efficient programming languages and HPC paradigms, MedGA may become a clinically feasible pre-processing step in real-time radiology applications. As a future extension of this work, in the case of large size images (e.g., 1000×1000 pixels) we plan to use GPUs, which represent an enabling technology for real-time radiology applications [126], since the running time of histograms computation can be considerably reduced by using a parallel implementation [385]. In the near future, we plan to apply MedGA as an image pre-processing phase also in other clinical contexts requiring MR image analysis and segmentation to provide useful insights for differential diagnosis and prognosis, such as in the case of breast cancer [189, 7] and meningiomas [179], also for differentiating tumor grade.

Final remarks

As a final remark, in Table 7.3 we summarize the best solutions for every problem discussed throughout this thesis.

Table 7.3 Summary of the best solutions to tackle the discussed problems in Life Sciences.

Problem	Input instance	Scenario	Solution	Architecture	References
Deterministic simulation	Small-scale model	Single simulation	LSODA - VODE	CPUs	[341] - [53]
		Multiple simulations	cupSODA - FiCoS	GPUs	[306] - [428]
	Large-scale model	Single simulation	LASSIE - FiCoS	GPUs	[425] - [428]
		Multiple simulations	FiCoS	GPUs	[428]
Stochastic simulation	Small-scale model	Single simulation	SSA	CPUs	[427]
		A few tens of simulations		MICs	[427]
		More than 80 simulations		GPUs	[427]
Parameter Estimation	Small-scale model	Single DTTS	FST-PSO + cupSODA	CPUs + GPUs	[311, 432]
		Multiple DTTS	MS^{2}PSO	CPUs + GPUs	[430]
	Large-scale model	Single DTTS	FST-PSO + FiCoS	CPUs + GPUs	[442]
Haplotype Assembly	Short reads	Low coverage	Exact approaches	CPUs	[342]
		High coverage	Probabilistic approches	CPUs	[124]
	Long reads	Low coverage	Exact approaches - GenHap	CPUs	[342] - [433, 431]
		High coverage	Probabilistic approches - GenHap	CPUs	[124] - [433, 431]
Image enhancement	Image with a nearly bimodal histogram	Single image	MedGA (single-core)	CPUs	[379, 377]
		Multiple images	MedGA (multi-cores)	CPUs	[379, 377]

According to the results shown in Chapter 4, in the case of deterministic simulations, different numerical integration methods can be exploited to solve the system of ODEs corresponding to the RBM under analysis. When the RBM is characterized by a few dozen species and reactions (small-scale models) and a single simulation is required, the CPU versions of LSODA and VODE represent the best choices. If multiple simulations have to be run, cupSODA should be preferred in order to perform in parallel all the simulations. In the case of large-scale models, LASSIE and FiCoS are the natural choices, when either a single or multiple simulations are taken into account, respectively. Concerning stochastic simulations of small-scale models, SSA can be effectively exploited. Even in this case, different architectures can be used to parallelize the number of simulations. For a single simulation, CPUs are capable of outperforming the other architectures; for a few tens of simulations, MICs achieve the best performance, whereas when more than 80 simulations are needed, GPUs should be considered.

Concerning the PE problem, as described in Chapter 5, different metaheuristics can represent effective approaches for its solution. However, when a single *Discrete-Time Target Series* (DTTS) is considered, FST-PSO coupled with cupSODA or FiCoS (depending on the size of the RBMs under analysis) achieves the best results, thus enabling the inference of the kinetic parameters that generate a simulated dynamics overlapping the DTTS. In the case of multiple DTTS, the proposed MS^{2}PSO is the natural choice, thanks to its capability of estimating a set of kinetic parameters that allows for obtaining simulated dynamics overlapping all the multiple DTTS. All these methodologies exploit both CPUs and GPUs to markedly decrease the required running time.

As reported in Chapters 1 and 6, several approaches have been proposed in the literature to deal with the Haplotype Assembly problem. Most of them have been designed to take into consideration only short reads and low coverage. In such a case, exact approaches (such as HapCol) should be used thanks to their ability to infer the correct pair of haplotypes. When the coverage is greater than 20-30$\times$, these approaches generally fail and probabilistic approaches (e.g., HapCUT2) are the best choice. GenHap can be effectively exploited to reconstruct the pair of haplotypes starting from long reads, generated by using third-generation sequencing technologies. The best performance of GenHap is achieved when dealing with data characterized by a high coverage, where most of the existing solutions fail due to the computational burden of this problem.

For what concerns the enhancement of biomedical images characterized by a nearly bimodal histogram, we proposed MedGA. According to the results shown in Chapter 7, MedGA is capable of outperforming the classical image enhancement techniques. Since it is the only method conceived for better emphasizing the two Gaussian distributions composing a bimodal histogram, it should be used to address this task. When a single image has to be processed, the single-core version is the correct choice, while in the case of a set of images, the multi-core version designed by using a Master-Slave approach can be used to reduce the required running time.

Bibliography

[1] Abdelbar, A. M., Abdelshahid, S., and Wunsch, D. C. (2005). Fuzzy PSO: a generalization of particle swarm optimization. In *Proc. IEEE International Joint Conference on Neural Networks*, volume 2, pages 1086–1091. IEEE.

[2] Abraham, A. and Liu, H. (2009). *Turbulent Particle Swarm Optimization Using Fuzzy Parameter Tuning*, volume 3, pages 291–312. Springer, Berlin Heidelberg, Germany.

[3] Ackermann, J., Baecher, P., Franzel, T., et al. (2009). Massively-parallel simulation of biochemical systems. In *Proc. Massively Parallel Computational Biology on GPUs, Jahrestagung der Gesellschaft für Informatik e.V*, volume 154 of *Lecture Notes in Computer Science*, pages 739–750.

[4] Ackley, D. H. (1987). *A Connectionist Machine for Genetic Hillclimbing*. Kluwer Academic Publishers, Norwell, MA, USA.

[5] Aguiar, D. and Istrail, S. (2013). Haplotype assembly in polyploid genomes and identical by descent shared tracts. *Bioinformatics*, 29(13):i352–i360.

[6] Ajay, S. S., Parker, S. C., Abaan, H. O., Fajardo, K. V. F., and Margulies, E. H. (2011). Accurate and comprehensive sequencing of personal genomes. *Genome Res.*

[7] Al-Najdawi, N., Biltawi, M., and Tedmori, S. (2015). Mammogram image visual enhancement, mass segmentation and classification. *Appl. Soft Comput.*, 35:175–185.

[8] Alba, E., Luque, G., and Nesmachnow, S. (2013). Parallel metaheuristics: Recent advances and new trends. *Int. Trans. Oper. Res.*, 20(1):1–48.

[9] Alba, E. and Tomassini, M. (2002). Parallelism and evolutionary algorithms. *IEEE Trans. Evol. Comput.*, 6(5):443–462.

[10] Alberghina, L. and Westerhoff, H. V. (2007). *Systems biology: Definitions and Perspectives*, volume 13. Springer, Berlin Heidelberg, Germany.

[11] Aldridge, B. B., Burke, J. M., Lauffenburger, D. A., and Sorger, P. K. (2006). Physicochemical modelling of cell signalling pathways. *Nat. Cell Biol.*, 8(11):1195–1203.

[12] Ali, M. Z., Awad, N. H., Suganthan, P. N., Shatnawi, A. M., and Reynolds, R. G. (2018). An improved class of real-coded genetic algorithms for numerical optimization. *Neurocomput.*, 275:155–166.

[13] Alves, J. M. and Posada, D. (2018). Sensitivity to sequencing depth in single-cell cancer genomics. *Genome Med.*, 10(1):29.

[14] Ambrosini, R. D., Wang, P., and O'Dell, W. G. (2010). Computer-aided detection of metastatic brain tumors using automated three-dimensional template matching. *J. Magn. Reson. Imaging*, 31(1):85–93.

[15] Anderson, D. P. (2004). BOINC: A system for public-resource computing and storage. In *Proc. Fifth IEEE/ACM International Workshop on Grid Computing*, pages 4–10. IEEE.

[16] Anderson, D. P. and Fedak, G. (2006). The computational and storage potential of volunteer computing. In *Proc. Sixth IEEE International Symposium on Cluster Computing and the Grid*, volume 1, pages 73–80. IEEE.

[17] Andrés, A. M., Clark, A. G., Shimmin, L., Boerwinkle, E., Sing, C. F., and Hixson, J. E. (2007). Understanding the accuracy of statistical haplotype inference with sequence data of known phase. *Genet. Epidemiol.*, 31(7):659–671.

[18] Andrews, S. S., Dinh, T., and Arkin, A. P. (2009). *Stochastic models of biological processes*, pages 8730–8749. Springer, New York, NY, USA.

[19] Armbrust, M., Fox, A., Griffith, R., et al. (2010). A View of Cloud Computing. *Commun. ACM*, 53(4):50–58.

[20] Arriaga-Garcia, E. F., Sanchez-Yanez, R. E., and Garcia-Hernandez, M. G. (2014). Image enhancement using bi-histogram equalization with adaptive sigmoid functions. In *Proc. IEEE International Conference on Electronics, Communications and Computers*, pages 28–34. IEEE.

[21] Arumugam, M. S. and Rao, M. V. C. (2008). 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. *Appl. Soft Comput.*, 8(1):324–336.

[22] Ault, S. and Holmgreen, E. (2009). Dynamics of the Brusselator. *Academia.*

[23] Baba, N. (1981). Convergence of a random optimization method for constrained optimization problems. *J. Optim. Theory Appl.*, 33(4):451–461.

[24] Back, T. (1994). Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Proc. First IEEE Conference on Evolutionary Computation*, volume 1, pages 57–62. IEEE.

[25] Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford University Press, Oxford, UK.

[26] Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. In *Proc. International Conference on Genetic Algorithms and their applications*, pages 101–111. Hillsdale, New Jersey.

[27] Bankman, I. N. (2009). *Part II - Segmentation*, pages 71–72. Academic Press, Burlington, MA, USA, 2 edition.

[28] Bansal, V. and Bafna, V. (2008). HapCUT: an efficient and accurate algorithm for the haplotype assembly problem. *Bioinformatics*, 24(16):i153–i159.

[29] Barabasi, A. L. and Oltvai, Z. N. (2004). Network biology: understanding the cell's functional organization. *Nat. Rev. Genet.*, 5(2):101.

[30] Bartels, R. H. and Golub, G. H. (1969). The simplex method of linear programming using LU decomposition. *Commun. ACM*, 12(5):266–268.

[31] Beccuti, M., Cazzaniga, P., Pennisi, M., Besozzi, D., Nobile, M. S., Pernice, S., Russo, G., Tangherloni, A., and Pappalardo, F. (2018). GPU accelerated analysis of Treg-Teff cross regulation in relapsing-remitting multiple sclerosis. In *Euro-Par 2018: Parallel Processing Workshops*, volume 11339 of *Lecture Notes in Computer Science*, pages 626–637. Springer.

[32] Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S., and Smith, K. (2011). Cython: the best of both worlds. *Comput. Sci. Eng.*, 13(2):31–39.

[33] Bellon, E. M., Haacke, E. M., Coleman, P. E., Sacco, D. C., Steiger, D. A., and Gangarosa, R. E. (1986). MR artifacts: a review. *Am. J. Roentgenol.*, 147(6):1271–1281.

[34] Ben-Elazar, S., Chor, B., and Yakhini, Z. (2016). Extending partial haplotypes to full genome haplotypes using chromosome conformation capture data. *Bioinformatics*, 32(17):i559–i566.

[35] Ben-Israel, A. (1966). A Newton-Raphson method for the solution of systems of equations. *J. Math. Anal. Appl.*, 15(2):243–252.

[36] Benedettini, S., Roli, A., and Di Gaspero, L. (2008). Two-level ACO for haplotype inference under pure parsimony. In *Proc. International Conference on Ant Colony Optimization and Swarm Intelligence*, volume 5217 of *Lecture Notes in Computer Science*, pages 179–190. Springer.

[37] Bennett, M. R., Pang, W. L., Ostroff, N. A., Baumgartner, B. L., Nayak, S., Tsimring, L. S., and Hasty, J. (2008). Metabolic gene regulation in a dynamically changing environment. *Nature*, 454(7208):1119.

[38] Beretta, S., Patterson, M. D., Zaccaria, S., Della Vedova, G., and Bonizzoni, P. (2018). HapCHAT: Adaptive haplotype assembly for efficiently leveraging high coverage in long reads. *BMC Bioinform.*, 19(1):252.

[39] Berger, E., Yorukoglu, D., Peng, J., and Berger, B. (2014). Haptree: A novel Bayesian framework for single individual polyplotyping using NGS data. *PLOS Comput. Biol.*, 10(3):e1003502.

[40] Bernaschi, M., Bisson, M., and Salvadore, F. (2014). Multi-Kepler GPU vs. multi-Intel MIC for spin systems simulations. *Comput. Phys. Commun.*, 185(10):2495–2503.

[41] Besozzi, D. (2016). Reaction-based models of biochemical networks. In Beckmann, A., Bienvenu, L., and Jonoska, N., editors, *Pursuit of the Universal*, volume 9709 of *Lecture Notes in Computer Science*, pages 24–34, Switzerland. Springer International Publishing.

[42] Besozzi, D., Cazzaniga, P., Mauri, G., and Pescini, D. (2010). BioSimWare: a software for the modeling, simulation and analysis of biological systems. In *Proc. International Conference on Membrane Computing*, pages 119–143. Springer.

[43] Besozzi, D., Cazzaniga, P., Mauri, G., Pescini, D., and Vanneschi, L. (2009). A comparison of genetic algorithms and particle swarm optimization for parameter estimation in stochastic biochemical systems. In Pizzuti, C., Ritchie, M. D., and Giacobini, M., editors, *Proc. European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, Lecture Notes in Computer Science, pages 116–127. Springer.

[44] Besozzi, D., Cazzaniga, P., Pescini, D., Mauri, G., Colombo, S., and Martegani, E. (2012). The role of feedback control mechanisms on the establishment of oscillatory regimes in the Ras/cAMP/PKA pathway in *s. cerevisiae*. *EURASIP J. Bioinform. Syst. Biol.*, 2012(1):10.

[45] Beyer, H. G. (2013). *The theory of Evolution Strategies*. Springer, Berlin Heidelberg, Germany.

[46] Beyer, H. G. and Schwefel, H. P. (2002). Evolution strategies–a comprehensive introduction. *Nat. Comput.*, 1(1):3–52.

[47] Bhandari, A. K., Kumar, A., Chaudhary, S., and Singh, G. (2016). A novel color image multilevel thresholding based segmentation using nature inspired optimization algorithms. *Expert Syst. Appl.*, 63:112–133.

[48] Bianchi, L. and Liò, P. (2016). Opportunities for community awareness platforms in personal genomics and bioinformatics education. *Brief. Bioinform.*, 18(6):1082–1090.

[49] Bianco, S., Ciocca, G., and Schettini, R. (2017). Combination of video change detection algorithms by genetic programming. *IEEE Trans. Evol. Comput.*, 21(6):914–928.

[50] Bland, A. S., Wells, J. C., Messer, O. E., et al. (2012). Titan: early experience with the Cray XK6 at Oak Ridge national laboratory. In *Proc. Cray User Group Conference*.

[51] Bonizzoni, P., Dondi, R., Klau, G. W., Pirola, Y., Pisanti, N., and Zaccaria, S. (2015). On the fixed parameter tractability and approximability of the minimum error correction problem. In *Proc.Annual Symposium on Combinatorial Pattern Matching*, volume 9133 of *Lecture Notes in Computer Science*, pages 100–113. Springer.

[52] Bracciali, A., Aldinucci, M., Patterson, M., Marschall, T., Pisanti, N., Merelli, I., and Torquati, M. (2016). pWhatsHap: efficient haplotyping for future generation sequencing. *BMC Bioinform.*, 17(Suppl 11):342.

[53] Brown, P. N., Byrne, G. D., and Hindmarsh, A. C. (1989). VODE: A variable-coefficient ODE solver. *SIAM J. Sci. Stat. Comp.*, 10(5):1038–1051.

[54] Brown, R. W., Cheng, Y. C. N., Haacke, E. M., Thompson, M. R., and Venkatesan, R. (2014). *Magnetic Resonance Imaging: Physical Principles and Sequence Design*. John Wiley & Sons, Hoboken, NJ, USA.

[55] Browning, S. R. and Browning, B. L. (2007). Rapid and accurate haplotype phasing and missing-data inference for whole-genome association studies by use of localized haplotype clustering. *Am. J. Hum. Genet.*, 81(5):1084–1097.

[56] Bruggeman, F. J. and Westerhoff, H. V. (2007). The nature of systems biology. *Trends Microbiol.*, 15(1):45–50.

[57] Burgard, A. P. and Maranas, C. D. (2001). Probing the performance limits of the Escherichia coli metabolic network subject to gene additions or deletions. *Biotechnol. Bioeng.*, 74(5):364–375.

[58] Burrage, K., Hegland, M., Macnamara, S., and Sidje, R. (2006). A Krylov-based finite state projection algorithm for solving the chemical master equation arising in the discrete modelling of biological systems. In Langville, A. N. and Stewart, W. J., editors, *Proc. Markov 150th Anniversary Conference*, pages 21–37.

[59] Butcher, J. C. (2008). *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, Chichester West Sussex, UK, 2 edition.

[60] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25(6):599–616.

[61] Cabitza, F., Rasoini, R., and Gensini, G. F. (2017). Unintended consequences of machine learning in medicine. *J. Am. Med. Assoc.*, 318(6):517–518.

[62] Cagnoni, S., Vanneschi, L., Azzini, A., and Tettamanzi, A. G. B. (2008). A critical assessment of some variants of particle swarm optimization. In *Proc. Workshops on Applications of Evolutionary Computation*, pages 565–574. Springer.

[63] Canny, J. (1986). A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, PAMI-8(6):679–698.

[64] Cao, Y., Gillespie, D. T., and Petzold, L. R. (2006). Efficient step size selection for the tau-leaping simulation method. *J. Chem. Phys.*, 124(4):044109.

[65] Carbonaro, A. and Zingaretti, P. (1999). A comprehensive approach to image-contrast enhancement. In *Proc. IEEE International Conference on Image Processing*, pages 241–246. IEEE.

[66] Carneiro, M. O., Russ, C., Ross, M. G., Gabriel, S. B., Nusbaum, C., and DePristo, M. A. (2012). Pacific Biosciences sequencing technology for genotyping and variation discovery in human data. *BMC Genomics*, 13(1):375.

[67] Cash, J. R. (2015). *Backward Differentiation Formulae*, pages 97–101. Springer, Berlin Heidelberg.

[68] Casper, J., Zweig, A. S., Villarreal, C., Tyner, C., Speir, M. L., Rosenbloom, K. R., Raney, B. J., Lee, C. M., Lee, B. T., Karolchik, D., et al. (2017). The UCSC Genome Browser database: 2018 update. *Nucleic Acids Res.*, 46(D1):D762–D769.

[69] Castelli, M., Vanneschi, L., and Silva, S. (2014). Prediction of the Unified Parkinson's Disease Rating Scale assessment using a genetic programming system with geometric semantic genetic operators. *Expert Syst. Appl.*, 41(10):4608–4616.

[70] Cazzaniga, P., Colombo, R., Nobile, M. S., Pescini, D., Mauri, G., and Besozzi, D. (2013). GPU-powered sensitivity analysis and parameter estimation of a reaction-based model of the post replication repair pathway in yeast. *TICSP SERIES*, 63:109–110.

[71] Cazzaniga, P., Damiani, C., Besozzi, D., Colombo, R., Nobile, M. S., Gaglio, D., Pescini, D., Molinari, S., Mauri, G., Alberghina, L., et al. (2014). Computational strategies for a system-level understanding of metabolism. *Metabolites*, 4(4):1034–1087.

[72] Cazzaniga, P., Ferrara, F., Nobile, M. S., et al. (2015a). Parallelizing biochemical stochastic simulations: a comparison of GPUs and Intel Xeon Phi processors. In Malyshkin, V., editor, *Proc. International Conference on Parallel Computing Technologies*, volume 9251 of *Lecture Notes in Computer Science*, pages 363–374.

[73] Cazzaniga, P., Nobile, M. S., and Besozzi, D. (2015b). The impact of particles initialization in PSO: Parameter estimation as a case in point. In *Proc. IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology*, pages 1–8. IEEE.

[74] Cazzaniga, P., Nobile, M. S., Tangherloni, A., and Besozzi, D. (2018). *Quantitative Biology: Theory, Computational Methods, and Models*, chapter Accelerating stochastic simulations of mechanistic models of biological systems: Advantages and issues in the parallelization on Graphics Processing Units. MIT Press, Cambridge, MA, USA.

[75] Cazzaniga, P., Pescini, D., Besozzi, D., Mauri, G., Colombo, S., and Martegani, E. (2008). Modeling and stochastic simulation of the Ras/cAMP/PKA pathway in the yeast *s. cerevisiae* evidences a key regulatory function for intracellular guanine nucleotides pools. *J. Biotechnol.*, 133(3):377–385.

[76] Chakraborty, M., Baldwin-Brown, J. G., Long, A. D., and Emerson, J. (2016). Contiguous and accurate de novo assembly of metazoan genomes with modest long read coverage. *Nucleic acids research*, 44(19):e147–e147.

[77] Chatterjee, A. and Siarry, P. (2006). Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization. *Comput. Oper. Res.*, 33(3):859–871.

[78] Chellaboina, V., Bhat, S. P., Haddad, W. M., and Bernstein, D. S. (2009). Modeling and analysis of mass-action kinetics. *IEEE Control Syst.*, 29(4):60–78.

[79] Chen, J., Yu, W., Tian, J., Chen, L., and Zhou, Z. (2017). Image contrast enhancement using an artificial bee colony algorithm. *Swarm Evol. Comput.*, In Press:1–8.

[80] Chen, S. D. and Ramli, A. R. (2003). Minimum mean brightness error bi-histogram equalization in contrast enhancement. *IEEE Trans. Consumer Electron.*, 49(4):1310–1319.

[81] Chen, Z. Z., Deng, F., and Wang, L. (2013). Exact algorithms for haplotype assembly from whole-genome sequence data. *Bioinformatics*, 29(16):1938–1945.

[82] Cho, Y. J., Ramakrishnan, N., and Cao, Y. (2008). Reconstructing chemical reaction networks: data mining meets system identification. In *Proc. 14th ACM International Conference on Knowledge Discovery and Data Mining*, pages 142–150. ACM.

[83] Choi, Y., Chan, A. P., Kirkness, E., Telenti, A., and Schork, N. J. (2018). Comparison of phasing strategies for whole human genomes. *PLOS Genet.*, 14(4):e1007308.

[84] Chou, I. C. and Voit, E. O. (2009a). Recent developments in parameter estimation and structure identification of biochemical and genomic systems. *Math. Biosci.*, 219(2):57–83.

[85] Chou, I. C. and Voit, E. O. (2009b). Recent developments in parameter estimation and structure identification of biochemical and genomic systems. *Math. Biosci.*, 219(2):57–83.

[86] Chrysos, G. (2014). Intel® Xeon Phi™ coprocessor-the architecture. *Intel Whitepaper*, 176.

[87] Chylek, L. A., Harris, L. A., Tung, C. S., Faeder, J. R., Lopez, C. F., and Hlavacek, W. S. (2014). Rule-based modeling: a computational approach for studying biomolecular site dynamics in cell signaling systems. *Wiley Interdiscip. Rev. Syst. Biol. Med.*, 6(1):13–36.

[88] Clerc, M. (2010). *Particle Swarm Optimization*, volume 93. John Wiley & Sons, Hoboken, NJ, USA.

[89] Cock, P. J. A., Fields, C. J., Goto, N., Heuer, M. L., and Rice, P. M. (2009). The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Res.*, 38(6):1767–1771.

[90] Compston, A. and Coles, A. (2008). Multiple sclerosis. *The Lancet*, 372(9648):1502–1517.

[91] Compston, G., McDonald, I., Noseworthy, J., Lassmann, H., Miller, D., Smith, K., Wekerle, H., and Confavreux, C. (2013). *McAlpine's Multiple Sclerosis*. Elsevier, Amsterdam, The Netherlands, 4 edition.

[92] Congress on Evolutionary Computation (CEC) (2017). Special Session & Competitions on Real-Parameter Single Objective Optimization. http://www.ntu.edu.sg/home/EPNSugan/index_files/CEC2017/CEC2017.htm. Online; Accessed on December 29, 2018.

[93] Cox, S. E., Haftka, R. T., Baker, C. A., Grossman, B., Mason, W. H., and Watson, L. T. (2001). A comparison of global optimization methods for the design of a high-speed civil transport. *J. Glob. Optim.*, 21(4):415–432.

[94] Craciun, G., Tang, Y., and Feinberg, M. (2006). Understanding bistability in complex enzyme-driven reaction networks. *Proc. Natl. Acad. Sci.*, 103(23):8697–8702.

[95] Cumbo, F., Nobile, M. S., Damiani, C., Colombo, R., Mauri, G., and Cazzaniga, P. (2017). COSYS: A computational infrastructure for systems biology. In Bracciali, A., Caravagna, G., Gilbert, D., and Tagliaferri, R., editors, *Proc. 13th International Meeting on Computational Intelligence Methods for Bioinformatics and Biostatistics*, volume 10477 of *Lecture Notes in Computer Science*, pages 82–92. Springer.

[96] Da Ros, S., Colusso, G., Weschenfelder, T. A., de Marsillac Terra, L., De Castilhos, F., Corazza, M. L., and Schwaab, M. (2013). A comparison among stochastic optimization algorithms for parameter estimation of biochemical kinetic models. *Appl. Soft Comput.*, 13(5):2205–2214.

[97] Daffner, R. H. (1980). Visual illusions in computed tomography: phenomena related to Mach effect. *Am. J. Roentgenol.*, 134(2):261–264.

[98] Dalcín, L., Paz, R., and Storti, M. (2005). MPI for Python. *J. Parallel Distrib. Comput.*, 65(9):1108–1115.

[99] Daly, M. J., Rioux, J. D., Schaffner, S. F., Hudson, T. J., and Lander, E. S. (2001). High-resolution haplotype structure in the human genome. *Nat. Genet.*, 29(2):229.

[100] Danecek, P., Auton, A., Abecasis, G., Albers, C. A., Banks, E., DePristo, M. A., Handsaker, R. E., Lunter, G., Marth, G. T., Sherry, S. T., et al. (2011). The variant call format and VCFtools. *Bioinformatics*, 27(15):2156–2158.

[101] Dantzig, G. B. and Thapa, M. N. (2006). *Linear Programming 1.* Springer, New York, NY, USA.

[102] Das, S. and Suganthan, P. N. (2011). Differential evolution: a survey of the state-of-the-art. *IEEE Trans. Evol. Comput.*, 15(1):4–31.

[103] Das, S. and Vikalo, H. (2015). SDhaP: haplotype assembly for diploids and polyploids via semi-definite programming. *BMC Genomics*, 16(1):260.

[104] de Araujo, A. F., Constantinou, C. E., and Tavares, J. M. R. S. (2014). New artificial life model for image enhancement. *Expert Syst. Appl.*, 41(13):5892–5906.

[105] De Jong, K. A. (2006). *Evolutionary Computation: A Unified Approach.* MIT press, Cambridge, MA, USA.

[106] De Myttenaere, A., Golden, B., Le Grand, B., and Rossi, F. (2016). Mean absolute percentage error for regression models. *Neurocomput.*, 192:38–48.

[107] De Oca, M. A. M., Stutzle, T., Birattari, M., and Dorigo, M. (2009). Frankenstein's PSO: a composite particle swarm optimization algorithm. *IEEE Trans. Evol. Comput.*, 13(5):1120–1132.

[108] Deb, K. and Jain, H. (2014). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, Part I: Solving problems with box constraints. *IEEE Trans. Evol. Computat.*, 18(4):577–601.

[109] Del Valle, Y., Venayagamoorthy, G. K., Mohagheghi, S., Harley, R. G., and Hernandez, J. C. (2008). Particle swarm optimization: basic concepts, variants and applications in power systems. *IEEE T. Evolut. Comput.*

[110] Delaneau, O., Marchini, J., and Zagury, J.-F. (2012). A linear complexity phasing method for thousands of genomes. *Nat. Methods*, 9(2):179.

[111] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7(Jan):1–30.

[112] Dieterich, J. M. and Hartke, B. (2012). Empirical review of standard benchmark functions using evolutionary global optimization. *Appl. Math.*, 3:1552–1564.

[113] Doerr, C., Ye, F., van Rijn, S., Wang, H., and Bäck, T. (2018). Towards a theory-guided benchmarking suite for discrete black-box optimization heuristics: profiling (1+ λ) EA variants on onemax and leadingones. In *Proc. Genetic and Evolutionary Computation Conference (GECCO)*, pages 951–958, New York, NY, USA. ACM.

[114] Dorigo, M., Bonabeau, E., and Theraulaz, G. (2000). Ant algorithms and stigmergy. *Future Gener. Comput. Syst.*, 16(8):851–871.

[115] Dormand, J. R. (1996). *Numerical Methods for Differential Equations: A Computational Approach*, volume 3. CRC Press, Boca Raton, FL, USA.

[116] Dormand, J. R. and Prince, P. J. (1980). A family of embedded Runge-Kutta formulae. *J. Comput. Appl. Math.*, 6(1):19–26.

[117] Draa, A. and Bouaziz, A. (2014). An artificial bee colony algorithm for image contrast enhancement. *Swarm Evol. Comput.*, 16:69–84.

[118] Dräger, A., Kronfeld, M., Ziller, M. J., Supper, J., Planatscher, H., Magnus, J. B., Oldiges, M., Kohlbacher, O., and Zell, A. (2009). Modeling metabolic networks in *C. glutamicum*: a comparison of rate laws in combination with various parameter optimization strategies. *BMC Syst. Biol.*, 3(1):5.

[119] Dubreuil, M., Gagné, C., and Parizeau, M. (2006). Analysis of a master-slave architecture for distributed evolutionary computations. *IEEE Trans. Syst. Man Cybern.*, 36(1):229–235.

[120] Duch, W. (2007). *What is Computational Intelligence and where is it going?*, pages 1–13. Springer, New York, NY, USA.

[121] Duitama, J., Huebsch, T., McEwen, G., Suk, E. K., and Hoehe, M. R. (2010). ReFHap: a reliable and fast algorithm for single individual haplotyping. In *Proc. First ACM International Conference on Bioinformatics and Computational Biology*, pages 160–169. ACM.

[122] Duncan, J. S. and Ayache, N. (2000). Medical image analysis: progress over two decades and the challenges ahead. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(1):85–106.

[123] Eberhart, R. and Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Proc. Sixth IEEE International Symposium on Micro Machine and Human Science*, pages 39–43. IEEE.

[124] Edge, P., Bafna, V., and Bansal, V. (2017). HapCUT2: robust and accurate haplotype assembly for diverse sequencing technologies. *Genome Res.*, 27(5):801–812.

[125] Eiben, A. E. and Smith, J. (2015). From evolutionary computation to the evolution of things. *Nature*, 521(7553):476–482.

[126] Eklund, A., Dufort, P., Forsberg, D., and LaConte, S. M. (2013). Medical image processing on the GPU–past, present and future. *Med. Image Anal.*, 17(8):1073–1094.

[127] Elowitz, M. B., Levine, A. J., Siggia, E. D., and Swain, P. S. (2002). Stochastic gene expression in a single cell. *Science*, 297(5584):1183–1186.

[128] Evans, P. M. (2008). Anatomical imaging for radiotherapy. *Phys. Med. Biol.*, 53(12):R151.

[129] Ewing, B., Hillier, L., Wendl, M. C., and Green, P. (1998). Base-calling of automated sequencer traces using Phred. I. Accuracy assessment. *Genome Res.*, 8(3):175–185.

[130] Fang, J., Varbanescu, A. L., Imbernon, B., Cecilia, J. M., and Perez-Sanchez, H. (2014). Parallel computation of non-bonded interactions in drug discovery: Nvidia GPUs vs. Intel Xeon Phi. In *Proc. 2nd International Work-Conference on Bioinformatics and Biomedical Engineering*, Lecture Notes in Computer Science.

[131] Fang, J., Varbanescu, A. L., Sips, H., Zhang, L., Che, Y., and Xu, C. (2013). Benchmarking Intel Xeon Phi to guide kernel design. Technical Report PDS-2013-005, Delft University of Technology.

[132] Farber, R. M. (2011). Topical perspective on massive threading and parallelism. *J. Mol. Graph. Model.*, 30:82–89.

[133] Fehlberg, E. (1968). Classical Fifth-, Sixth-, Seventh-, And Eighth-order Runge-Kutta Formulas With Stepsize. Technical Report R-287, NASA.

[134] Fehlberg, E. (1969). Low-order classical Runge-Kutta formulas with stepsize control and their application to some heat transfer problems. Technical report, NASA.

[135] Filonenko, E. and Seeram, E. (2018). Big data: The next era of informatics and data science in medical imaging: A literature review. *J. Clin. Exp. Radiol.*, 1.

[136] Fletcher, R. (2013). *Practical Methods of Optimization*. John Wiley & Sons, Hoboken, NJ, USA, 2 edition.

[137] Fontenot, J. D. and Rudensky, A. Y. (2005). A well adapted regulatory contrivance: Regulatory T cell development and the forkhead family transcription factor Foxp3. *Nat. Immunol.*, 6(4):331–337.

[138] Fortin, F., De Rainville, F., Gardner, M., Parizeau, M., and Gagné, C. (2012). DEAP: Evolutionary algorithms made easy. *J. Mach. Learn. Res.*, 13:2171–2175.

[139] Foster, I. and Kesselman, C. (2003). *The Grid 2: Blueprint for a New Computing Infrastructure*. Elsevier, Amsterdam, The Netherlands.

[140] Gabriel, S. B., Schaffner, S. F., Nguyen, H., Moore, J. M., Roy, J., Blumenstiel, B., Higgins, J., DeFelice, M., Lochner, A., Faggart, M., et al. (2002). The structure of haplotype blocks in the human genome. *Science*, 296(5576):2225–2229.

[141] Gallagher, M. (2016). Towards improved benchmarking of black-box optimization algorithms using clustering problems. *Soft Comput.*, 20(10):3835–3849.

[142] Gämperle, R., Müller, S. D., and Koumoutsakos, P. (2002). A parameter study for differential evolution. In *Proc. Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, pages 293–298. WSEAS Press.

[143] Gan, H. S., Swee, T. T., Abdul Karim, A. H., Sayuti, K. A., Abdul Kadir, M. R., Tham, W. K., Wong, L. X., Chaudhary, K. T., Ali, J., and Yupapin, P. P. (2014). Medical image visual appearance improvement using bihistogram Bezier curve contrast enhancement: Data from the osteoarthritis initiative. *Sci. World J.*, 2014(294104):1–13.

[144] Gandhamal, A., Talbar, S., Gajre, S., Hani, A. F. M., and Kumar, D. (2017). Local gray level S-curve transformation–a generalized contrast enhancement technique for medical images. *Comput. Biol. Med*, 83:120–133.

[145] García, S., Molina, D., Lozano, M., and Herrera, F. (2009). A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization. *J. Heuristics*, 15(6):617—-644.

[146] García-Nieto, J. and Alba, E. (2011). Restart particle swarm optimization with velocity modulation: a scalability test. *Soft Comput.*, 15(11):2221–2232.

[147] Gear, C. W. (1968). The control of parameters in the automatic integration of ordinary differential equations. *Int. Rep.*, 757.

[148] Genetic and Evolutionary Computation Conference (GECCO) (2018). GECCO Workshop on Real-Parameter Black-Box Optimization Benchmarking (BBOB). http://numbbo.github.io/workshops/BBOB-2018/. Online; Accessed on December 29, 2018.

[149] Genohub (2018). Recommended coverage and read depth for NGS applications. https://genohub.com/recommended-sequencing-coverage-by-application. Online; Accessed on December 29, 2018.

[150] Gerstenberger, R., Besta, M., and Hoefler, T. (2014). Enabling highly-scalable remote memory access programming with MPI-3 one sided. *Sci. Programming*, 22(2):75–91.

[151] Ghaemmaghami, S., Huh, W. K., Bower, K., Howson, R. W., Belle, A., Dephoure, N., O'shea, E. K., and Weissman, J. S. (2003). Global analysis of protein expression in yeast. *Nature*, 425(6959):737.

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

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

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

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

[156] Gillespie, D. T. (2000). The chemical Langevin equation. *J. Chem. Phys.*, 113(1):297–306.

[157] Gillespie, D. T. (2001). Approximate accelerated stochastic simulation of chemically reacting systems. *J. Phys. Chem.*, 115(4):1716–1733.

[158] Gillespie, D. T. (2007). Stochastic simulation of chemical kinetics. *Annu. Rev. Phys. Chem.*, 58:35–55.

[159] Gillespie, D. T. and Petzold, L. R. (2003). Improved leap-size selection for accelerated stochastic simulation. *J. Chem. Phys.*, 119:8229–8234.

[160] Gillies, R. J., Kinahan, P. E., and Hricak, H. (2015). Radiomics: images are more than pictures, they are data. *Radiology*, 278(2):563–577.

[161] Glover, F. (1989). Tabu search—part I. *ORSA J. Comput.*, 1(3):190–206.

[162] Glover, F. (1990). Tabu search—part II. *ORSA J. Comput.*, 2(1):4–32.

[163] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Boston, MA, USA, 1 edition.

[164] Gong, Y. J., Chen, W. N., Zhan, Z. H., Zhang, J., Li, Y., Zhang, Q., and Li, J. J. (2015). Distributed evolutionary algorithms and their models: A survey of the state-of-the-art. *Appl. Soft Comput.*, 34:286–300.

[165] Gonzalez, R. C. and Woods, R. E. (2002). *Digital Image Processing*. Prentice Hall Inc., Upper Saddle River, NJ, USA, 3 edition.

[166] Goodin, D. S. (2009). The causal cascade to multiple sclerosis: A model for MS pathogenesis. *PLOS One*, 4(2).

[167] Granville, V., Krivánek, M., and Rasson, J. P. (1994). Simulated annealing: A proof of convergence. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(6):652–656.

[168] Gray, K. (2003). *Microsoft DirectX 9 Programmable Graphics Pipeline*. Microsoft Press, Redmond, WA, USA.

[169] Greenberg, H. J., Hart, W. E., and Lancia, G. (2004). Opportunities for combinatorial optimization in computational biology. *Informs. J. Comput.*, 16(3):211–231.

[170] Greene, C. S., Tan, J., Ung, M., Moore, J. H., and Cheng, C. (2014). Big data bioinformatics. *J. Cell. Physiol.*, 229(12):1896–1900.

[171] Gropp, W., Lusk, E., Doss, N., and Skjellum, A. (1996). A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Comput.*, 22(6):789–828.

[172] Gropp, W. D., Gropp, W., Lusk, E., and Skjellum, A. (1999). *Using MPI: Portable Parallel Programming With the Message-Passing Interface*, volume 1. MIT press, Cambridge, MA, USA.

[173] Gunawan, R., Cao, Y., Petzold, L., and Doyle III, F. J. (2005). Sensitivity analysis of discrete stochastic systems. *Biophys. J.*, 88(4):2530–2540.

[174] Guo, S., Diep, D., Plongthongkum, N., Fung, H. L., Zhang, K., and Zhang, K. (2017). Identification of methylation haplotype blocks aids in deconvolution of heterogeneous tissue samples and tumor tissue-of-origin mapping from plasma DNA. *Nat. Genet.*, 49(4):635–642.

[175] Gutjahr, W. J. (2002). ACO algorithms with guaranteed convergence to the optimal solution. *Inf. Process Lett.*, 82(3):145–153.

[176] Hairer, E., Nørsett, S. P., and Wanner, G. (1993). *Solving Ordinary Differential Equations I*. Springer, Berlin Heidelberg, Germany.

[177] Hairer, E. and Wanner, G. (1996). *Solving Ordinary Differential Equations II*. Springer, Berlin Heidelberg, Germany.

[178] Hairer, E. and Wanner, G. (1999). Stiff differential equations solved by Radau methods. *J. Comput. Appl. Math.*, 111(1-2):93–111.

[179] Hale, A. T., Wang, L., Strother, M. K., and Chambless, L. B. (2018). Differentiating meningioma grade by imaging features on magnetic resonance imaging. *J. Clin. Neurosci.*, 48:71–75.

[180] Hall, E. L. (1974). Almost uniform distributions for computer image enhancement. *IEEE Trans. Comput.*, 100(2):207–208.

[181] Halyo, V., LeGresley, P., Lujan, P., Karpusenko, V., and Vladimirov, A. (2014). First evaluation of the CPU, GPGPU and MIC architectures for real time particle tracking based on Hough transform at the LHC. *J. Instrum.*, 9(04).

[182] Hansen, N., Arnold, D. V., and Auger, A. (2015). *Evolution strategies*, pages 871–898. Springer, Berlin Heidelberg.

[183] Hansen, N., Auger, A., Mersmann, O., Tusar, T., and Brockhoff, D. (2016). COCO: A platform for comparing continuous optimizers in a black-box setting. *arXiv preprint arXiv:1603.08785*.

[184] Hansen, N. and Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proc. IEEE International Conference on Evolutionary Computation*, pages 312–317. IEEE.

[185] Haque, A., Engel, J., Teichmann, S. A., and Lönnberg, T. (2017). A practical guide to single-cell RNA-sequencing for biomedical research and clinical applications. *Genome Med.*, 9(1):75.

[186] Harris, L. A., Hogg, J. S., Tapia, J.-J., Sekar, J. A., Gupta, S., Korsunsky, I., Arora, A., Barua, D., Sheehan, R. P., and Faeder, J. R. (2016). BioNetGen 2.2: advances in rule-based modeling. *Bioinformatics*, 32(21):3366–3368.

[187] Harrison, K. R., Engelbrecht, A. P., and Ombuki-Berman, B. M. (2016). Inertia weight control strategies for particle swarm optimization. *Swarm Intell.*, 10(4):267–305.

[188] Hashemi, S., Kiani, S., Noroozi, N., and Moghaddam, M. E. (2010). An image contrast enhancement method based on genetic algorithm. *Pattern Recogn. Lett.*, 31(13):1816–1824.

[189] Hassanien, A. E., Moftah, H. M., Azar, A. T., and Shoman, M. (2014). MRI breast cancer diagnosis hybrid approach using adaptive ant-based segmentation and multilayer perceptron neural networks classifier. *Appl. Soft Comput.*, 14(Part A):62–71.

[190] Hauschild, M. and Pelikan, M. (2011). An introduction and survey of estimation of distribution algorithms. *Swarm Evol. Comput.*, 1(3):111–128.

[191] He, D., Choi, A., Pipatsrisawat, K., Darwiche, A., and Eskin, E. (2010). Optimal algorithms for haplotype assembly from whole-genome sequence data. *Bioinformatics*, 26(12):i183–i190.

[192] Hellander, A. (2008). Efficient computation of transient solutions of the chemical master equation based on uniformization and quasi-Monte Carlo. *J. Chem. Phys.*, 128(15):154109.

[193] Hermisson, J. and Pennings, P. S. (2017). Soft sweeps and beyond: understanding the patterns and probabilities of selection footprints under rapid adaptation. *Methods Ecol. Evol.*, 8(6):700–716.

[194] Higham, D. J. and Trefethen, L. N. (1993). Stiffness of ODEs. *BIT Numer. Math.*, 33(2):285–303.

[195] Hirschhorn, J. N. and Daly, M. J. (2005). Genome-wide association studies for common diseases and complex traits. *Nat. Rev. Genet.*, 6(2):95.

[196] Hlavacek, W. S., Faeder, J. R., Blinov, M. L., Perelson, A. S., and Goldstein, B. (2003). The complexity of complexes in signal transduction. *Biotechnol. Bioeng.*, 84(7):783–794.

[197] Hlavacek, W. S., Faeder, J. R., Blinov, M. L., Posner, R. G., Hucka, M., and Fontana, W. (2006). Rules for modeling signal-transduction systems. *Sci. STKE.*, 2006(344):re6–re6.

[198] Hoberock, J. and Bell, N. (2010). Thrust: A parallel template library. Version 1.7.0.

[199] Hoefler, T., Lumsdaine, A., and Dongarra, J. (2009). Towards efficient mapreduce using MPI. In *Proc. European Parallel Virtual Machine/Message Passing Interface*, pages 240–249. Springer.

[200] Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA.

[201] Hölldobler, B. and Wilson, E. O. (2008). *The Superorganism: The Beauty, Elegance, and Strangeness of Insect Societies*. WW Norton & Company.

[202] Hoops, S., Sahle, S., Gauges, R., et al. (2006). COPASI - a COmplex PAthway SImulator. *Bioinformatics*, 22(24):3067–3074.

[203] Hu, Z., Xiong, S., Su, Q., and Zhang, X. (2013). Sufficient conditions for global convergence of differential evolution algorithm. *J. Appl. Math.*, 2013.

[204] Hwang, B., Lee, J. H., and Bang, D. (2018). Single-cell RNA sequencing technologies and bioinformatics pipelines. *Exp. Mol. Med.*, 50(8):96.

[205] Ideker, T., Galitski, T., and Hood, L. (2001). A new approach to decoding life: Systems biology. *Annu. Rev. Genomics. Hum. Genet.*, 2(1):343–372.

[206] Ilie, S. and Bădică, C. (2013). Multi-agent distributed framework for swarm intelligence. *Procedia Computer Sci.*, 18:611–620.

[207] Inc., P. (1988). The renderman interface specification, version 3.0.

[208] Iosup, A. and Epema, D. (2011). Grid computing workloads. *IEEE Internet Comput.*, 15(2):19–26.

[209] Ismail, M. A. (2004). Parallel genetic algorithms (PGAs): master slave paradigm approach using MPI. In *Proc. E-Tech 2004*, pages 83–87. IEEE.

[210] Jackson, K. R. (1991). A survey of parallel numerical methods for initial value problems for ordinary differential equations. *IEEE Trans. Magn.*, 27(5):3792–3797.

[211] Jahnke, T. and Huisinga, W. (2007). Solving the chemical master equation for monomolecular reaction systems analytically. *J. Math. Biol.*, 54(1):1–26.

[212] Jain, M., Fiddes, I. T., Miga, K. H., Olsen, H. E., Paten, B., and Akeson, M. (2015). Improved data analysis for the MinION Nanopore sequencer. *Nat. Methods*, 12(4):351.

[213] Jain, M., Koren, S., Miga, K. H., Quick, J., Rand, A. C., Sasani, T. A., Tyson, J. R., Beggs, A. D., Dilthey, A. T., Fiddes, I. T., et al. (2018). Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nat. Biotechnol.*, 36(4):338.

[214] Jain, M., Olsen, H. E., Paten, B., and Akeson, M. (2016). The Oxford Nanopore MinION: delivery of nanopore sequencing to the genomics community. *Genome Biol.*, 17(1):239.

[215] Jamil, M. and Yang, X.-S. (2013). A literature survey of benchmark functions for global optimization problems. *Int. J. Math. Model. Num. Opt.*, 4(2):150–194.

[216] Jamshidi, N. and Palsson, B. Ø. (2010). Mass action stoichiometric simulation models: incorporating kinetics and regulation into stoichiometric models. *Biophys. J.*, 98(2):175–185.

[217] Jeffers, J. and Reinders, J. (2013). *Intel Xeon Phi Coprocessor High Performance Programming*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition.

[218] João, A., Gambaruto, A. M., Tiago, J., and Sequeira, A. (2016). Computational advances applied to medical image processing: an update. *Open Access Bioinform.*, 2016(8):1–15.

[219] Jones, E., Oliphant, T., Peterson, P., et al. (2014). SciPy: Open source scientific tools for Python.

[220] Joubert, W., Archibald, R., Berrill, M., et al. (2015). Accelerated application development: The ORNL Titan experience. *Comput. Electr. Eng.*, 46:123–138.

[221] Kämpf, J. H., Wetter, M., and Robinson, D. (2010). A comparison of global optimization algorithms with standard benchmark functions and real-world applications using EnergyPlus. *J. Build. Perf. Sim.*, 3(2):103–120.

[222] Karaboga, D. and Akay, B. (2009). A comparative study of artificial bee colony algorithm. *Appl. Math. Comput.*, 214(1):108–132.

[223] Karaboga, D. and Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *J. Glob. Optim.*, 39(3):459–471.

[224] Kennedy, J. (2006). *Swarm intelligence*, pages 187–219. Springer.

[225] Kennedy, J. and Eberhart, R. C. (1995). Particle swarm optimization. In *Proc. IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE.

[226] Kent, E., Hoops, S., and Mendes, P. (2012). Condor-COPASI: high-throughput computing for biochemical networks. *BMC Syst. Biol.*, 6(1):91.

[227] Kim, Y. T. (1997). Contrast enhancement using brightness preserving bi-histogram equalization. *IEEE Trans. Consumer Electron.*, 43(1):1–8.

[228] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

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

[230] Kittler, J. and Illingworth, J. (1986). Minimum error thresholding. *Pattern Recognit.*, 19(1):41–47.

[231] Klee, V. and Minty, G. J. (1970). How good is the simplex algorithm. Technical report, Department of Mathematics, University of Washington.

[232] Kline, M. P. and Morimoto, R. I. (1997). Repression of the heat shock factor 1 transcriptional activation domain is modulated by constitutive phosphorylation. *Mol. Cell. Biol.*, 17(4):2107–2115.

[233] Kohmura, H. and Wakahara, T. (2006). Determining optimal filters for binarization of degraded characters in color using genetic algorithms. In *Proc. 18th IEEE International Conference on Pattern Recognition*, volume 3, pages 661–664. IEEE.

[234] Komarov, I. and D'Souza, R. M. (2012). Accelerating the Gillespie exact stochastic simulation algorithm using hybrid parallel execution on graphics processing units. *PLOS One*, 7(11):e46693.

[235] Komarov, I., D'Souza, R. M., and Tapia, J. (2012). Accelerating the Gillespie τ-leaping method using graphics processing units. *PLOS One*, 7(6):e37370.

[236] Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA, 1 edition.

[237] Kraus, J., Pivanti, M., Schifano, S. F., Tripiccione, R., and Zanella, M. (2013). Benchmarking GPUs with a parallel Lattice-Boltzmann code. In *Proc. 25th IEEE International Symposium on Computer Architecture and High Performance Computing*, pages 160–167. IEEE.

[238] Krupinski, E. A. (2010). Current perspectives in medical image perception. *Atten. Percept. Psychophys.*, 72(5):1205–1217.

[239] Kuleshov, V. (2014). Probabilistic single-individual haplotyping. *Bioinformatics*, 30(17):i379–i385.

[240] Kuleshov, V., Xie, D., Chen, R., Pushkarev, D., Ma, Z., Blauwkamp, T., Kertesz, M., and Snyder, M. (2014). Whole-genome haplotyping using long reads and statistical methods. *Nat. Biotech.*, 32(3):261–266.

[241] Kurita, T., Otsu, N., and Abdelmalek, N. (1992). Maximum likelihood thresholding based on population mixture models. *Pattern Recognit.*, 25(10):1231–1240.

[242] Kwok, S. M., Chandrasekhar, R., Attikiouzel, Y., and Rickard, M. T. (2004). Automatic pectoral muscle segmentation on mediolateral oblique view mammograms. *IEEE Trans. Med. Imaging*, 23(9):1129–1140.

[243] Lambin, P., Leijenaar, R. T. H., Deist, T. M., Peerlings, J., de Jong, E. E. C., van Timmeren, J., Sanduleanu, S., Larue, R. T. H. M., Even, A. J. G., Jochems, A., et al. (2017). Radiomics: the bridge between medical imaging and personalized medicine. *Nat. Rev. Clin. Oncol.*, 14(12):749–762.

[244] Lameter, C. (2013). NUMA (Non-Uniform Memory Access): an overview. *Queue*, 11(7):40–51.

[245] Lander, E. S. and Waterman, M. S. (1988). Genomic mapping by fingerprinting random clones: a mathematical analysis. *Genomics*, 2(3):231–239.

[246] Langdon, W. B. and Poli, R. (2007). Evolving problems to learn about particle swarm optimizers and other search algorithms. *IEEE Trans. Evol. Comput.*, 11(5):561–578.

[247] Larrañaga, P. and Lozano, J. A. (2001). *A New Tool for Evolutionary Computation: Estimation of Distribution Algorithms*, volume 2. Springer, New York, NY, USA.

[248] Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T. (1979). Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Softw.*, 5(3):308–323.

[249] L'Ecuyer, P. (1996). Combined multiple recursive random number generators. *Oper. Res.*, 44(5):816–822.

[250] L'Ecuyer, P., Simard, R., Chen, E. J., and Kelton, W. D. (2002). An object-oriented random-number package with many long streams and substreams. *Oper. Res.*, 50(6):1073–1075.

[251] Leksell, L. (1951). The stereotaxic method and radiosurgery of the brain. *Acta Chir. Scand.*, 102(4):316–319.

[252] Lepock, J. R., Frey, H. E., and Ritchie, K. P. (1993). Protein denaturation in intact hepatocytes and isolated cellular organelles during heat shock. *J. Cell Biol.*, 122(6):1267–1276.

[253] Leu, M. S. and Yeh, M. F. (2012). Grey particle swarm optimization. *Appl. Soft Comp.*, 12(9):2985–2996.

[254] Levy, S., Sutton, G., Ng, P. C., Feuk, L., Halpern, A. L., Walenz, B. P., Axelrod, N., Huang, J., Kirkness, E. F., Denisov, G., et al. (2007). The diploid genome sequence of an individual human. *PLOS Biol.*, 5(10):e254.

[255] Li, C., Nguyen, T. T., Zeng, S., Yang, M., and Wu, M. (2018). An open framework for constructing continuous optimization problems. *IEEE Trans. Cybern.*, pages 1–15.

[256] Li, C. and Yang, S. (2008). A generalized approach to construct benchmark problems for dynamic optimization. In Li, X. et al., editors, *Simulated Evolution and Learning (SEAL)*, Lecture Notes in Computer Science, pages 391–400. Springer.

[257] Li, C., Yang, Y., Xiao, L., Li, Y., Zhou, Y., and Zhao, J. (2016). A novel image enhancement method using fuzzy Sure entropy. *Neurocomput.*, 215:196–211.

[258] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., and Durbin, R. (2009). The sequence alignment/map format and SAMtools. *Bioinformatics*, 25(16):2078–2079.

[259] Li, H. and Petzold, L. R. (2010). Efficient parallelization of the stochastic simulation algorithm for chemically reacting systems on the Graphics Processing Unit. *Int. J. High Perform. C.*, 24(2):107–116.

[260] Li, Y. and Xul, Z. (2003). An ant colony optimization heuristic for solving maximum independent set problems. In *Proc. Fifth IEEE International Conference on Computational Intelligence and Multimedia Applications*, pages 206–211. IEEE.

[261] Liang, J. J., Suganthan, P. N., and Deb, K. (2005). Novel composition test functions for numerical global optimization. In *Proc. Swarm Intelligence Symposium (SIS)*, pages 68–75. IEEE.

[262] Light, S., Kraulis, P., and Elofsson, A. (2005). Preferential attachment in the evolution of metabolic networks. *BMC Genomics*, 6(1):159.

[263] Lillacci, G. and Khammash, M. (2010). Parameter estimation and model selection in computational biology. *PLOS Comput. Biol.*, 6(3):e1000696.

[264] Limpert, E., Stahel, W. A., and Abbt, M. (2001). Log-normal distributions across the sciences: Keys and clues. *BioScience*, 51(5):341–352.

[265] Lippert, R., Schwartz, R., Lancia, G., and Istrail, S. (2002). Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem. *Brief. Bioinform.*, 3(1):23–31.

[266] Loh, P. R., Danecek, P., Palamara, P. F., Fuchsberger, C., Reshef, Y. A., Finucane, H. K., Schoenherr, S., Forer, L., McCarthy, S., Abecasis, G. R., et al. (2016). Reference-based phasing using the haplotype reference consortium panel. *Nature Genet.*, 48(11):1443.

[267] Lou, Y. and Yuen, S. Y. (2018). On constructing alternative benchmark suite for evolutionary algorithms. *Swarm Evol. Comput.*

[268] Lou, Y., Yuen, S. Y., and Chen, G. (2018). Evolving benchmark functions using Kruskal-Wallis test. In *Proc. Genetic and Evolutionary Computation Conference Companion (GECCO)*, pages 1337–1341. ACM.

[269] Lund, J. M., Hsing, L., Pham, T. T., and Rudensky, A. Y. (2008). Coordination of early protective immunity to viral infection by regulatory T cells. *Science*, 320(5880):1220–1224.

[270] Luo, C., Tsementzi, D., Kyrpides, N., Read, T., and Konstantinidis, K. T. (2012). Direct comparisons of Illumina vs. Roche 454 sequencing technologies on the same microbial community DNA sample. *PLOS One*, 7(2):e30087.

[271] Lyakh, D. I. (2015). An efficient tensor transpose algorithm for multicore CPU, Intel Xeon Phi, and NVidia Tesla GPU. *Comput. Phys. Commun.*, 189:84–91.

[272] Macchiarulo, L. (2008). A massively parallel implementation of Gillespie algorithm on FPGAs. In *Proc. IEEE International Conference on Engineering in Medicine and Biology Society*, pages 1343–1346. IEEE.

[273] Macready, W. G. and Wolpert, D. H. (1996). What makes an optimization problem hard? *Complexity*, 1(5):40–46.

[274] Maisto, D., Donnarumma, F., and Pezzulo, G. (2015). Divide et impera: subgoaling reduces the complexity of probabilistic inference and problem solving. *J. R. Soc. Interface*, 12(104):20141335.

[275] Malan, K. M. and Engelbrecht, A. P. (2013). A survey of techniques for characterising fitness landscapes and some possible ways forward. *Inf. Sci.*, 241:148–163.

[276] Mark, W. R., Glanville, R. S., Akeley, K., and Kilgard, M. J. (2003). CG: a system for programming graphics hardware in a C-like language. *ACM Trans. Graph.*, 22(3):896–907.

[277] Mathews, J. H. and Fink, K. D. (1998). *Numerical methods using MATLAB*. Prentice-Hall Inc., Upper Saddle River, NJ, USA, 3 edition.

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

[279] Matyas, J. (1965). Random optimization. *Autom. Rem. Contr.*, 26(2):246–253.

[280] Maus, C., Rybacki, S., and Uhrmacher, A. M. (2011). Rule-based multi-level modeling of cell biological systems. *BMC Syst. Biol.*, 5(1):166.

[281] McElroy, K. E., Luciani, F., and Thomas, T. (2012). GemSIM: general, error-model based simulator of next-generation sequencing data. *BMC Genomics*, 13(1):74.

[282] Medina-Carnicer, R., Muñoz-Salinas, R., Carmona-Poyato, A., and Madrid-Cuevas, F. J. (2011). A novel histogram transformation to improve the performance of thresholding methods in edge detection. *Pattern Recognit. Lett.*, 32(5):676–693.

[283] Meier, R., Knecht, U., Loosli, T., Bauer, S., Slotboom, J., Wiest, R., and Reyes, M. (2016). Clinical evaluation of a fully-automatic segmentation method for longitudinal brain tumor volumetry. *Sci. Rep.*, 6.

[284] Mendes, P. (2001). *Modeling large biochemical systems from functional genomic data: parameter estimation*, pages 163–168. MIT Press, Cambridge, MA, USA.

[285] Mendes, P. and Kell, D. (1998). Non-linear optimization of biochemical pathways: applications to metabolic engineering and parameter estimation. *Bioinformatics*, 14(10):869–883.

[286] Merelli, I., Liò, P., and Milanesi, L. (2013). Nuchart: an R package to study gene spatial neighbourhoods with multi-omics annotations. *PLOS One*, 8(9):e75146.

[287] Metallo, C. M. and Vander Heiden, M. G. (2013). Understanding metabolic regulation and its influence on cell physiology. *Mol. Cell.*, 49(3):388–398.

[288] Metcalfe, P., Liney, G. P., Holloway, L., Walker, A., Barton, M., Delaney, G. P., Vinod, S., and Tome, W. (2013). The potential for an enhanced role for MRI in radiation-therapy treatment planning. *Technol. Cancer. Res. Treat.*, 12(5):429–446.

[289] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21(6):1087–1092.

[290] Mezura-Montes, E., Velázquez-Reyes, J., and Coello Coello, C. A. (2006). A comparative study of differential evolution variants for global optimization. In *Proc. 8th ACM Annual Conference on Genetic and evolutionary computation*, pages 485–492. ACM.

[291] Militello, C., Rundo, L., and Gilardi, M. C. (2014). Applications of imaging processing to MRgFUS treatment for fibroids: a review. *Transl. Cancer Res.*, 3(5):472–482.

[292] Militello, C., Vitabile, S., Rundo, L., Russo, G., Midiri, M., and Gilardi, M. C. (2015). A fully automatic 2D segmentation method for uterine fibroid in MRgFUS treatment evaluation. *Comput. Biol. Med.*, 62:277–292.

[293] Miller, B. L. and Goldberg, D. E. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex Syst.*, 9(3):193–212.

[294] Moles, C. G., Mendes, P., and Banga, J. R. (2003). Parameter estimation in biochemical pathways: a comparison of global optimization methods. *Genome Res.*, 13(11):2467–2474.

[295] Mühlenbein, H. (2012). *Convergence theorems of estimation of distribution algorithms*, pages 91–108. Springer, Berlin Heidelberg.

[296] Munsky, B. and Khammash, M. (2006). The finite state projection algorithm for the solution of the chemical master equation. *J. Chem. Phys.*, 124(4):044104.

[297] Munteanu, C. and Rosa, A. (2004). Gray-scale image enhancement as an automatic process driven by evolution. *IEEE Trans. Syst. Man Cybern.*, 34(2):1292–1298.

[298] Na, J. C., Lee, J.-C., Rhee, J.-K., and Shin, S.-Y. (2018). PEATH: Single individual haplotyping by a probabilistic evolutionary algorithm with toggling. *Bioinformatics*, page bty012.

[299] Nachman, M. W. (2001). Single nucleotide polymorphisms and recombination rate in humans. *Trends Genet.*, 17(9):481–485.

[300] Nakamura, K., Yoshida, R., Nagasaki, M., Miyano, S., and Higuchi, T. (2009). Parameter estimation of *in silico* biological pathways with particle filtering towards a petascale computing. In *Biocomputing 2009*, pages 227–238. World Scientific.

[301] Nelson, D. L., Lehninger, A. L., and Cox, M. M. (2008). *Lehninger principles of biochemistry*. Macmillan, London, UK, 5 edition.

[302] Nickolls, J. and Dally, W. J. (2010). The GPU computing era. *IEEE micro*, 30(2):56–69.

[303] Nobile, M. S., Besozzi, D., Cazzaniga, P., G., M., and Pescini, D. (2012a). Estimating reaction constants in stochastic biological systems with a multi-swarm PSO running on GPUs. In Soule, T., editor, *Proc. 14th ACM International Conference. on Genetic and Evolutionary Computation*, pages 1421–1422. ACM.

[304] Nobile, M. S., Besozzi, D., Cazzaniga, P., G., M., and Pescini, D. (2012b). A GPU-based multi-swarm PSO method for parameter estimation in stochastic biological systems exploiting discrete-time target series. In Giacobini, M., Vanneschi, L., and Bush, W. S., editors, *Proc. European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, volume 7246 of *Lecture Notes in Computer Science*, pages 74–85. Springer.

[305] Nobile, M. S., Cazzaniga, P., Besozzi, D., Colombo, R., Mauri, G., and Pasi, G. (2018a). Fuzzy self-tuning PSO: A settings-free algorithm for global optimization. *Swarm Evol. Comput.*, 39:70–85.

[306] Nobile, M. S., Cazzaniga, P., Besozzi, D., and Mauri, G. (2014a). GPU-accelerated simulations of mass-action kinetics models with cupSODA. *J. Supercomput.*, 69(1):17–24.

[307] Nobile, M. S., Cazzaniga, P., Besozzi, D., Pescini, D., and Mauri, G. (2014b). cu-TauLeaping: a gpu-powered tau-leaping stochastic simulator for massive parallel analyses of biological systems. *PLOS One*, 9(3):e91963.

[308] Nobile, M. S., Cazzaniga, P., Tangherloni, A., and Besozzi, D. (2017). Graphics processing units in bioinformatics, computational biology and systems biology. *Brief. Bioinformatics*, 18(5):870–885.

[309] Nobile, M. S., Pasi, G., Cazzaniga, P., Besozzi, D., Colombo, R., and Mauri, G. (2015). Proactive particles in swarm optimization: a self-tuning algorithm based on fuzzy logic. In *Proc. IEEE International Conference on Fuzzy Systems*, pages 1–8. IEEE.

[310] Nobile, M. S., Tangherloni, A., Besozzi, D., and Cazzaniga, P. (2016). GPU-powered and settings-free parameter estimation of biochemical systems. In *Proc. IEEE Congress on Evolutionary Computation*, pages 32–39. IEEE.

[311] Nobile, M. S., Tangherloni, A., Rundo, L., Spolaor, S., Besozzi, D., Mauri, G., and Cazzaniga, P. (2018b). Computational intelligence for parameter estimation of biochemical systems. In *Proc. IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE.

[312] Nvidia (2012). Nvidia CUDA C programming guide v5.0.

[313] Nvidia (2014). NVBIO webpage. http://nvlabs.github.io/nvbio/. Online; Accessed on December 29, 2018.

[314] Nvidia (2015a). cuBLAS library 7.5.

[315] Nvidia (2015b). cuBLAS library 8.0.

[316] Nvidia (2015a). cuFFT library user's guide 7.5.

[317] Nvidia (2015b). CURAND library programming guide 7.5.

[318] Nvidia (2015a). cuSPARSE library 7.5.

[319] Nvidia (2015b). Nvidia CUDA C programming guide 7.5.

[320] Nvidia (2015c). Nvidia performance primitives 7.5.

[321] Nvidia (2016a). CUDA C programming guide, version 8.0.

[322] Nvidia (2016b). Nvidia Tesla P100.

[323] Orellana, A. and Minetti, G. F. (2009). A modified binary-PSO for continuous optimization. In *Proc. XV Congreso Argentino de Ciencias de la Computación*.

[324] Orth, J. D., Thiele, I., and Palsson, B. Ø. (2010). What is flux balance analysis? *Nat. Biotech.*, 28(3):245.

[325] Ortiz, A., Górriz, J. M., Ramírez, J., Salas-Gonzalez, D., and Llamas-Elvira, J. M. (2013). Two fully-unsupervised methods for MR brain image segmentation using SOM-based strategies. *Appl. Soft Comput.*, 13(5):2668–2682.

[326] Otsu, N. (1975). A threshold selection method from gray-level histograms. *IEEE Trans. Syst. Man Cybern.*, 11(285-296):23–27.

[327] Otto, S. P. and Whitton, J. (2000). Polyploid incidence and evolution. *Annu. Rev. Genet.*, 34(1):401–437.

[328] O'Brien, J., Kla, K. M., Hopkins, I. B., Malecki, E. A., and McKenna, M. C. (2007). Kinetic parameters and lactate dehydrogenase isozyme activities support possible lactate utilization by neurons. *Neurochem. Res.*, 32(4-5):597–607.

[329] PacBio (2014). Data release: $\sim 54\times$ long-read coverage for PacBio-only de novo human genome assembly.

[330] Pacheco, P. S. (1997). *Parallel Programming with MPI*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[331] Paranjape, R. B. (2009). *Fundamental Enhancement Techniques*, pages 3–18. Academic Press, Burlington, MA, USA, 2 edition.

[332] Pasupuleti, S. and Battiti, R. O. (2006). The gregarious particle swarm optimizer (G-PSO). In *Proc. 8th ACM Annual Conference on Genetic and Evolutionary Computation*, pages 67–74. ACM.

[333] Patterson, M., Marschall, T., Pisanti, N., Van Iersel, L., Stougie, L., Klau, G. W., and Schönhuth, A. (2015). WhatsHap: weighted haplotype assembly for future-generation sequencing reads. *J. Comput. Biol.*, 22(6):498–509.

[334] Paulinas, M. and Ušinskas, A. (2007). A survey of genetic algorithms applications for image enhancement and segmentation. *Inf. Technol. Control*, 36(3):278—-284.

[335] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Inc., San Francisco, CA, USA.

[336] Penas, D. R., González, P., Egea, J. A., Doallo, R., and Banga, J. R. (2017). Parameter estimation in large-scale systems biology models: a parallel and self-adaptive cooperative strategy. *BMC Bioinform.*, 18(1):52.

[337] Pennisi, M., Rajput, A.-M., Toldo, L., and Pappalardo, F. (2013). Agent based modeling of Treg-Teff cross regulation in relapsing-remitting multiple sclerosis. *BMC Bioinform.*, 14(Suppl 16):S9.

[338] Peper, A., Grimbergen, C. A., Spaan, J. A. E., Souren, J. E. M., and Wijk, R. V. (1998). A mathematical model of the hsp70 regulation in the cell. *Int. J. Hyperthermia*, 14(1):97–124.

[339] Pescini, D., Cazzaniga, P., Besozzi, D., Mauri, G., Amigoni, L., Colombo, S., and Martegani, E. (2012). Simulation of the Ras/cAMP/PKA pathway in budding yeast highlights the establishment of stable oscillatory states. *Biotechnol. Adv.*, 30(1):99–107.

[340] Petre, I., Mizera, A., Hyder, C. L., Meinander, A., Mikhailov, A., Morimoto, R. I., Sistonen, L., Eriksson, J. E., and Back, R. J. (2011). A simple mass-action model for the eukaryotic heat shock response and its mathematical validation. *Nat. Comput.*, 10(1):595–612.

[341] Petzold, L. (1983). Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations. *SIAM J. Sci. Stat. Comp.*, 4(1):136–148.

[342] Pirola, Y., Zaccaria, S., Dondi, R., Klau, G. W., Pisanti, N., and Bonizzoni, P. (2015). HapCol: accurate and memory-efficient haplotype assembly from long reads. *Bioinformatics*, 32(11):1610–1617.

[343] Poli, R. and Cagnoni, S. (1997). Genetic programming with user-driven selection: experiments on the evolution of algorithms for image enhancement. In *Proc. 2nd Annual Conference on Genetic Programming*, pages 269–277.

[344] Poli, R., Kennedy, J., and Blackwell, T. (2007). Particle swarm optimization. *Swarm Intell.*, 1(1):33–57.

[345] Pomerening, J. R. (2008). Uncovering mechanisms of bistability in biological systems. *Curr. Opin. Biotechnol.*, 19(4):381–388.

[346] Ponsonby, A. L., van der Mei, I., Dwyer, T., Blizzard, L., Taylor, B., Kemp, A., Simmons, R., and Kilpatrick, T. (2005). Exposure to infant siblings during early life and risk of multiple sclerosis. *JAMA*, 293(4):463–469.

[347] Poovathingal, S. K. and Gunawan, R. (2010). Global parameter estimation methods for stochastic biochemical systems. *BMC Bioinform.*, 11(1):414.

[348] Price, N. D., Papin, J. A., and Palsson, B. Ø. (2002). Determination of redundancy and systems properties of the metabolic network of Helicobacter pylori using genome-scale extreme pathway analysis. *Genome Res.*, 12(5):760–769.

[349] Qin, A. K., Huang, V. L., and Suganthan, P. N. (2009). Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE T. Evolut. Comput.*, 13(2):398–417.

[350] Quail, M. A., Kozarewa, I., Smith, F., Scally, A., Stephens, P. J., Durbin, R., Swerdlow, H., and Turner, D. J. (2008). A large genome center's improvements to the Illumina sequencing system. *Nat. Methods*, 5(12):1005.

[351] Ramakrishna, R., Edwards, J. S., McCulloch, A., and Palsson, B. Ø. (2001). Flux-balance analysis of mitochondrial energy metabolism: consequences of systemic stoichiometric constraints. *Am. J. Physiol. Regul. Integr. Comp. Physiol.*, 280(3):R695–R704.

[352] Rasmussen, R. V. and Trick, M. A. (2008). Round robin scheduling–a survey. *Eur. J. Oper. Res.*, 188(3):617–636.

[353] Ratliff, F. (1984). Why Mach bands are not seen at the edges of a step. *Vision Res.*, 24(2):163–165.

[354] Rechenberg, I. (1973). *Evolution Strategy: Optimization of Technical Systems by Means of Biological Evolution.* Fromman-Holzboog, Stuttgart, Germany.

[355] Reinhardt, J. A., Baltrus, D. A., Nishimura, M. T., Jeck, W. R., Jones, C. D., and Dangl, J. L. (2009). De novo assembly using low-coverage short read sequence data from the rice pathogen *Pseudomonas syringae* pv. *oryzae*. *Genome Res.*, 19(2):294–305.

[356] Reinker, S., Altman, R. M., and Timmer, J. (2006). Parameter estimation in stochastic biochemical reactions. *IEE P. Syst. Biol.*, 153(4):168–178.

[357] Rezaee Jordehi, A. and Jasni, J. (2013). Parameter selection in particle swarm optimisation: a survey. *J. Exp. Theor. Artif. Intell.*, 25(4):527–542.

[358] Rhoads, A. and Au, K. F. (2015). PacBio sequencing and its applications. *Genomics Proteomics Bioinformatics*, 13(5):278–289.

[359] Ridler, T. W. and Calvard, S. (1978). Picture thresholding using an iterative selection method. *IEEE Trans. Syst. Man Cybern.*, 8(8):630–632.

[360] Rimmer, A., Phan, H., Mathieson, I., Iqbal, Z., Twigg, S. R. F., Wilkie, A. O. M., McVean, G., Lunter, G., WGS500 Consortium, et al. (2014). Integrating mapping-, assembly-and haplotype-based approaches for calling variants in clinical sequencing applications. *Nat Genet*, 46(8):912.

[361] Riva, S. G., Tangherloni, A., Nobile, M. S., Cazzaniga, P., and Besozzi, D. (2018). SMGen: A novel generator of synthetic models of biological systems. *BMC Bioinform.* Under Preparation.

[362] Roberts, A. (2008). Magnetic resonance-guided focused ultrasound for uterine fibroids. *Semin. Interv. Radiol.*, 25(4):394.

[363] Roberts, R. J., Carneiro, M. O., and Schatz, M. C. (2013). The advantages of SMRT sequencing. *Genome Biol.*, 14(6):405.

[364] Rodriguez, F. and Arkhipova, I. R. (2018). Transposable elements and polyploid evolution in animals. *Curr. Opin. Genet. Dev.*, 49:115–123.

[365] Rogowska, J. (2009). *Overview and Fundamentals of Medical Image Segmentation*, pages 73–90. Academic Press, Burlington, MA, USA, 2 edition.

[366] Romero, J. and Cotta, C. (2005). Optimization by island-structured decentralized particle swarms. In Reusch, B., editor, *Computational Intelligence, Theory and Applications*, volume 33 of *Advances in Soft Computing*, pages 25–33. Springer.

[367] Rueckert, D., Glocker, B., and Kainz, B. (2016). Learning clinically useful information from images: Past, present and future. *Med. Image Anal.*, 33:13–18.

[368] Rumschinski, P., Borchers, S., Bosio, S., Weismantel, R., and Findeisen, R. (2010). Set-base dynamical parameter estimation and model invalidation for biochemical reaction networks. *BMC Syst. Biol.*, 4(1):69.

[369] Runarsson, T. P. and Yao, X. (2000). Stochastic ranking for constrained evolutionary optimization. *IEEE Trans. Evol. Comput.*, 4(3):284–294.

[370] Rundo, L., Han, C., Nagano, Y., Zhang, J., Hataya, R., Militello, C., et al. (2018a). USE-Net: Incorporating squeeze-and-excitation blocks into U-Net for prostate zonal segmentation of multi-institutional MRI datasets. *Neurocomput.* Submitted.

[371] Rundo, L., Han, C., Zhang, J., Hataya, R., Nagano, Y., Militello, C., et al. (2018b). CNN-based prostate zonal segmentation on T2-weighted MR images: a cross-dataset study. In *28th Italian Workshop on Neural Networks (WIRN) 2018, Vietri sul Mare (SA), Italy, June 13-15, 2018*, Smart Innovation, Systems and Technologies. Springer. (Accepted).

[372] Rundo, L., Militello, C., Russo, G., Vitabile, S., Gilardi, M. C., and Mauri, G. (2018c). GTVcut for neuro-radiosurgery treatment planning: an MRI brain cancer seeded image segmentation method based on a cellular automata model. *Nat. Comput.*, 17(3):521–536.

[373] Rundo, L., Militello, C., Tangherloni, A., Russo, G., Lagalla, R., Mauri, G., Gilardi, M. C., and Vitabile, S. (2019). Computer-assisted approaches for uterine fibroid segmentation in MRgFUS treatments: quantitative evaluation and clinical feasibility analysis. In *Quantifying and Processing Biomedical and Behavioral Signals*, volume 103 of *Smart Innovation, Systems and Technologies*, pages 229–241. Springer.

[374] Rundo, L., Militello, C., Tangherloni, A., Russo, G., Vitabile, S., Gilardi, M. C., and Mauri, G. (2018d). NeXt for neuro-radiosurgery: A fully automatic approach for necrosis extraction in brain tumor MRI using an unsupervised machine learning technique. *Int. J. Imag. Syst. Tech.*, 28(1):21–37.

[375] Rundo, L., Militello, C., Vitabile, S., Casarino, C., Russo, G., Midiri, M., and Gilardi, M. C. (2016a). Combining split-and-merge and multi-seed region growing algorithms for uterine fibroid segmentation in MRgFUS treatments. *Med. Biol. Eng. Comput.*, 54(7):1071–1084.

[376] Rundo, L., Stefano, A., Militello, C., Russo, G., Sabini, M. G., D'Arrigo, C., Marletta, F., Ippolito, M., Mauri, G., Vitabile, S., and Gilardi, M. C. (2017). A fully automatic approach for multimodal PET and MR image segmentation in Gamma Knife treatment planning. *Comput. Methods Programs Biomed.*, 144:77–96.

[377] Rundo, L., Tangherloni, A., Cazzaniga, P., Nobile, M. S., Russo G., Gilardi, M. C., Vitabile, S., Mauri, G., Besozzi, D., and Militello, C. (2018e). Genetic algorithms improve thresholding-based segmentation of bimodal magnetic resonance images. *Comput. Methods Programs Biomed.* Submitted.

[378] Rundo, L., Tangherloni, A., Militello, C., Gilardi, M. C., and Mauri, G. (2016b). Multimodal medical image registration using particle swarm optimization: a review. In *Proc. IEEE Symposium Series on Computational Intelligence*, pages 1–8. IEEE.

[379] Rundo, L., Tangherloni, A., Nobile, M. S., Militello, C., Besozzi, D., Mauri, G., and Cazzaniga, P. (2018f). MedGA: A novel evolutionary method for image enhancement in medical imaging systems. *Expert Syst. Appl.*, 119:387–399.

[380] Rundo, L., Tangherloni, A., Tyson, D. R., Betta, R., Nobile, M. S., Spolaor, S., Militello, C., Lubbock, A. L. R., Mauri, G., Quaranta, V., Besozzi, D., Lopez, C. F., and Cazzaniga, P. (2018g). ACDC: Automated cell detection and counting for time-lapse fluorescence microscopy. *Med. Biol. Eng. Comput.* Under Preparation.

[381] Saitoh, F. (1999). Image contrast enhancement using genetic algorithm. In *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, volume 4, pages 899–904. IEEE.

[382] Sakaguchi, S., Sakaguchi, N., Asano, M., Itoh, M., and Toda, M. (1995). Immunologic self-tolerance maintained by activated T cells expressing IL-2 receptor alpha-chains (CD25). breakdown of a single mechanism of self-tolerance causes various autoimmune diseases. *J. Immunol.*, 155(3):1152–1164.

[383] Santangelo, G. M. (2006). Glucose signaling in *saccharomyces cerevisiae*. *Microbiol. Mol. Biol. Rev.*, 70(1):253–282.

[384] Sawai, H. and Kizu, S. (1998). Parameter-free genetic algorithm inspired by "disparity theory of evolution". In *Proc. International Conference on Parallel Problem Solving from Nature*, pages 702–711. Springer.

[385] Scheuermann, T. and Hensley, J. (2007). Efficient histogram generation using scattering on GPUs. In *Proc. ACM Symposium on Interactive 3D Graphics and Games*, pages 33–37. ACM.

[386] Schilling, C. H., Letscher, D., and Palsson, B. Ø. (2000). Theory for the systemic definition of metabolic pathways and their use in interpreting metabolic function from a pathway-oriented perspective. *J. Theor. Biol.*, 203(3):229–248.

[387] Schomburg, I., Chang, A., Ebeling, C., Gremse, M., Heldt, C., Huhn, G., and Schomburg, D. (2004). BRENDA, the enzyme database: updates and major new developments. *Nucleic Acids Res.*, 32(suppl_1):D431–D433.

[388] Schwaab, M., Biscaia Jr, E. C., Monteiro, J. L., and Pinto, J. C. (2008). Nonlinear parameter estimation through particle swarm optimization. *Chem. Eng. Sci.*, 63(6):1542–1552.

[389] Sebag, M. and Ducoulombier, A. (1998). Extending population-based incremental learning to continuous search spaces. In *Proc. International Conference on Parallel Problem Solving from Nature*, pages 418–427. Springer.

[390] Seeley, T. D. (1989). The honey bee colony as a superorganism. *Am. Sci.*, 77(6):546–553.

[391] Senol Cali, D., Kim, J. S., Ghose, S., Alkan, C., and Mutlu, O. (2018). Nanopore sequencing technology and tools for genome assembly: computational analysis of the current state, bottlenecks and future directions. *Brief. Bioinform.* bby017.

[392] Shanmugavadivu, P. and Balasubramanian, K. (2014). Particle swarm optimized multi-objective histogram equalization for image enhancement. *Opt. Laser Technol.*, 57:243–251.

[393] Sheehan, S. and Song, Y. S. (2016). Deep learning for population genetic inference. *PLOS Comput. Biol.*, 12(3):e1004845.

[394] Shi, Y. and Eberhart, R. C. (2001). Fuzzy adaptive particle swarm optimization. In *Proc. IEEE Congress on Evolutionary Computation*, volume 1, pages 101–106. IEEE.

[395] Shimoda, T., Suzuki, S., Ohue, M., et al. (2015a). Protein-protein docking on hardware accelerators: comparison of GPU and MIC architectures. *BMC Syst. Biol.*, 9(Suppl 1):S6.

[396] Shimoda, T., Suzuki, S., Ohue, M., Ishida, T., and Akiyama, Y. (2015b). Protein-protein docking on hardware accelerators: comparison of GPU and MIC architectures. *BMC Syst. Biol.*, 9(Suppl 1):S6.

[397] Sidje, R. B., Burrage, K., and MacNamara, S. (2007). Inexact uniformization method for computing transient distributions of Markov chains. *SIAM J. Sci. Stat. Comp.*, 29(6):2562–2580.

[398] Sims, D., Sudbery, I., Ilott, N. E., Heger, A., and Ponting, C. P. (2014). Sequencing depth and coverage: key considerations in genomic analyses. *Nat. Rev. Genet.*, 15(2):121.

[399] Singh, M., Verma, A., and Sharma, N. (2017). Bat optimization based neuron model of stochastic resonance for the enhancement of MR images. *Biocybern. Biomed. Eng.*, 37(1):124–134.

[400] Sled, J. G., Zijdenbos, A. P., and Evans, A. C. (1998). A nonparametric method for automatic correction of intensity nonuniformity in MRI data. *IEEE Trans. Med. Imaging*, 17(1):87–97.

[401] Smallbone, K., Messiha, H. L., Carroll, K. M., Winder, C. L., Malys, N., Dunn, W. B., Murabito, E., Swainston, N., Dada, J. O., Khan, F., et al. (2013). A model of yeast glycolysis based on a consistent kinetic characterisation of all its enzymes. *FEBS Lett.*, 587(17):2832–2841.

[402] Smith, T. B. (2010). MRI artifacts and correction strategies. *Imaging Med.*, 2(4):445–457.

[403] Smooke, M. D. (1983). Error estimate for the modified Newton method with applications to the solution of nonlinear, two-point boundary-value problems. *J. Optim. Theory Appl.*, 39(4):489–511.

[404] Snyder, M. W., Adey, A., Kitzman, J. O., and Shendure, J. (2015). Haplotype-resolved genome sequencing: experimental methods and applications. *Nat. Rev. Genet.*, 16(6):344–358.

[405] Solis, F. J. and Wets, R. J. B. (1981). Minimization by random search techniques. *Math. Oper. Res.*, 6(1):19–30.

[406] Sonka, M., Hlavac, V., and Boyle, R. (2014). *Image Processing, Analysis, and Machine Vision*. Cengage Learning, Boston, MA, USA, 4 edition.

[407] Soon, W. W., Hariharan, M., and Snyder, M. P. (2013). High-throughput sequencing for biology and medicine. *Mol. Syst. Biol.*, 9(1):640.

[408] Sospedra, M. and Martin, R. (2005). Immunology of multiple sclerosis. *Annu. Rev. Immunol.*, 23(May):683–747.

[409] Spolaor, S., Tangherloni, A., Rundo, L., Nobile, M. S., and Cazzaniga, P. (2017). Reboot strategies in particle swarm optimization and their impact on parameter estimation of biochemical systems. In *Proc. IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology*, pages 1–8. IEEE.

[410] Spolaor, S., Tangherloni, A., Rundo, L., Nobile, M. S., and Cazzaniga, P. (2019). Estimation of kinetic reaction constants: Exploiting reboot strategies to improve pso's performance. In *Proc. 14th International Meeting on Computational Intelligence Methods for Bioinformatics and Biostatistics*, Lecture Notes in Bioinformatics. Springer. In Press.

[411] Starck, J. L., Murtagh, F., Candes, E. J., and Donoho, D. L. (2003). Gray and color image contrast enhancement by the curvelet transform. *IEEE Trans. Image Process.*, 12(6):706–717.

[412] Stewart, W. J. (1994). *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, Princeton, NJ, USA.

[413] Storn, R. and Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *J. Global Optim.*, 11(4):341–359.

[414] Streets, A. M. and Huang, Y. (2014). How deep is enough in single-cell RNA-seq? *Nat. Biotechnol.*, 32(10):1005.

[415] Stützle, T. and Dorigo, M. (1999). ACO algorithms for the traveling salesman problem. In Miettinen, K., Neittaanmäki, P. Mäkelä, M. M., and Periaux, J., editors, *Evolutionary algorithms in engineering and computer science*, pages 163–183. Wiley.

[416] Stutzle, T. and Dorigo, M. (2002). A short convergence proof for a class of ant colony optimization algorithms. *IEEE Trans. Evol. Comput.*, 6(4):358–365.

[417] Styner, M., Brechbuhler, C., Széckely, G., and Gerig, G. (2000). Parametric estimate of intensity inhomogeneities applied to MRI. *IEEE Trans. Med. Imaging*, 19(3):153–165.

[418] Sugeno, M. (1985). *Industrial Applications of Fuzzy Control*. Elsevier Science Inc., New York, NY, USA.

[419] Sumiyoshi, K., Hirata, K., Hiroi, N., et al. (2015). Acceleration of discrete stochastic biochemical simulation using GPGPU. *Front. Physiol.*, 6(42).

[420] Sundström, P., Juto, P., Wadell, G., Hallmans, G., Svenningsson, A., Nyström, L., Dillner, J., and Forsgren, L. (2004). An altered immune response to Epstein-Barr virus in multiple sclerosis. *Neurology*, 62(12):2277–2282.

[421] Sur, S., Koop, M. J., and Panda, D. K. (2006). High-performance and scalable MPI over InfiniBand with reduced memory usage: an in-depth performance analysis. In *Proc. ACM/IEEE conference on Supercomputing*, page 105. ACM.

[422] Szymańska, P., Martin, K. R., MacKeigan, J. P., Hlavacek, W. S., and Lipniacki, T. (2015). Computational analysis of an Autophagy/Translation switch based on mutual inhibition of MTORC1 and ULK1. *PLOS One*, 10(3):e0116550.

[423] Taha, A. A. and Hanbury, A. (2015). Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool. *BMC Med. imaging*, 15(1):29.

[424] Taherkhani, M. and Safabakhsh, R. (2016). A novel stability-based adaptive inertia weight for particle swarm optimization. *Appl. Soft Comp.*, 38:281–295.

[425] Tangherloni, A., Nobile, M. S., Besozzi, D., Mauri, G., and Cazzaniga, P. (2017a). LASSIE: simulating large-scale models of biochemical systems on GPUs. *BMC Bioinform.*, 18(1):246.

[426] Tangherloni, A., Nobile, M. S., and Cazzaniga, P. (2016). GPU-powered bat algorithm for the parameter estimation of biochemical kinetic values. In *Proc. IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology*, pages 1–6. IEEE.

[427] Tangherloni, A., Nobile, M. S., Cazzaniga, P., Besozzi, D., and Mauri, G. (2017b). Gillespie's stochastic simulation algorithm on mic coprocessors. *J. Supercomput.*, 73(2):676–686.

[428] Tangherloni, A., Nobile, M. S., Cazzaniga, P., Capitoli, G., Spolaor, S., Rundo, L., Mauri, G., and D., B. (2018a). FiCoS: a fine- and coarse-grained gpu-powered deterministic simulator for biochemical networks. *Sci. Rep.* Under Preparation.

[429] Tangherloni, A., Rundo, L., and Nobile, M. S. (2017c). Proactive particles in swarm optimization: a settings-free algorithm for real-parameter single objective optimization problems. In *Proc. IEEE Congress on Evolutionary Computation*, pages 1940–1947. IEEE.

[430] Tangherloni, A., Rundo, L., Spolaor, S., Cazzaniga, P., and Nobile, M. S. (2018b). GPU-powered multi-swarm parameter estimation of biological systems: A master-slave approach. In *Proc. 26th IEEE Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pages 698–705. IEEE.

[431] Tangherloni, A., Rundo, L., Spolaor, S., Nobile, M. S., Merelli, I., Besozzi, D., Mauri, G., Cazzaniga, P., and Liò, P. (2018c). High performance computing for haplotyping: Models and platforms. In *Euro-Par 2018: Parallel Processing Workshops*, volume 11339 of *Lecture Notes in Computer Science*, pages 650–661. Springer.

[432] Tangherloni, A., Spolaor, S., Cazzaniga, P., Besozzi, D., Rundo, L., Mauri, G., and Nobile, M. S. (2018d). Benchmark functions do not capture the complexity of biochemical parameter estimation. *Appl. Soft Comput.* Submitted.

[433] Tangherloni, A., Spolaor, S., Rundo, L., Nobile, M. S., Cazzaniga, P., Mauri, G., Liò, P., Merelli, I., and Besozzi, D. (2018e). GenHap: A novel computational method based on genetic algorithms for haplotype assembly. *BMC Bioinform.* In press.

[434] Tanweer, M. R., Suresh, S., and Sundararajan, N. (2015). Self regulating particle swarm optimization algorithm. *Inf. Sci.*, 294:182–202.

[435] Tashkova, K., Korošec, P., Šilc, J., Todorovski, L., and Džeroski, S. (2011). Parameter estimation with bio-inspired meta-heuristic optimization: modeling the dynamics of endocytosis. *BMC Syst. Biol.*, 5(159).

[436] Theraulaz, G. and Bonabeau, E. (1999). A brief history of stigmergy. *Artif. Life.*, 5(2):97–116.

[437] Thiagarajan, S. U., Congdon, C., Naik, S., et al. (2013). Intel Xeon Phi Coprocessor Developer's Quick Start Guide.

[438] Thohura, S. and Rahman, A. (2013). Numerical approach for solving stiff differential equations: A comparative study. *J. Sci. Front. Res. Math. Decision Sci.*, 13:7–18.

[439] Tian, D. and Li, N. (2009). Fuzzy particle swarm optimization algorithm. In *Proc. International Joint Conference on Artificial Intelligence*, pages 263–267, Pasadena, CA.

[440] Tian, T. and Burrage, K. (2005). Parallel implementation of stochastic simulation of large-scale cellular processes. In *Proc. 8th International Conference on High-Performance Computing in Asia-Pacific Region*, pages 621–626.

[441] Toennies, K. D. (2017). *Guide to Medical Image Analysis*. Springer, London, UK, 3 edition.

[442] Totis, N., Tangherloni, A., Beccuti, M., Cazzaniga, P., Nobile, M. S., Besozzi, D., Pennisi, M., and Pappalardo, F. (2018). GPU powered parameter estimation of a large-scale kinetic metabolic model. In *15th International Meeting on Computational Intelligence Methods for Bioinformatics and Biostatistics*.

[443] Trelea, I. C. (2003). The particle swarm optimization algorithm: convergence analysis and parameter selection. *Inf. Process. Lett.*, 85(6):317–325.

[444] Trinh, C. T., Wlaschin, A., and Srienc, F. (2009). Elementary mode analysis: a useful metabolic pathway analysis tool for characterizing cellular metabolism. *Appl. Microbiol. Biotechnol.*, 81(5):813.

[445] Trussell, H. J. (1979). Comments on picture thresholding using an iterative selection method. *IEEE Trans. Syst. Man Cybern.*, 9(5):311–311.

[446] Uhlén, M., Fagerberg, L., Hallström, B. M., Lindskog, C., Oksvold, P., Mardinoglu, A., Sivertsson, Å., Kampf, C., Sjöstedt, E., Asplund, A., et al. (2015). Tissue-based map of the human proteome. *Science*, 347(6220):1260419.

[447] Vaidyanathan, S. (2015). Dynamics and control of Brusselator chemical reaction. *Int. J. ChemTech. Res.*, 8(6):740–749.

[448] Van den Bergh, F. and Engelbrecht, A. P. (2006). A study of particle swarm optimization particle trajectories. *Inf. Sci.*, 176(8):937–971.

[449] Van Kampen, N. G. (2007). *Stochastic Processes in Physics and Chemistry*. Elsevier, Amsterdam, The Netherlands, 3 edition.

[450] Vélez De Mendizábal, N., Carneiro, J., Solé, R. V., Goñi, J., Bragard, J., Martinez-Forero, I., Martinez-Pasamar, S., Sepulcre, J., Torrealdea, J., Bagnato, F., Garcia-Ojalvo, J., and Villoslada, P. (2011). Modeling the effector-regulatory T cell cross-regulation reveals the intrinsic character of relapses in Multiple Sclerosis. *BMC Syst. Biol.*, 5(114).

[451] Vellela, M. and Qian, H. (2008). Stochastic dynamics and non-equilibrium thermodynamics of a bistable chemical system: the Schlögl model revisited. *J. Royal Soc. Interface*, pages rsif–2008.

[452] Verkauf, B. S. (1993). Changing trends in treatment of leiomyomata uteri. *Curr. Opin. Obstet. Gynecol.*, 5(3):301–310.

[453] Vitorino, L. N., Ribeiro, S. F., and Bastos-Filho, C. J. A. (2015). A mechanism based on artificial bee colony to generate diversity in particle swarm optimization. *Neurocomput.*, 148:39–45.

[454] Voit, E. O. (2000). *Computational Analysis of Biochemical Systems: A Practical Guide for Biochemists and Molecular Biologists*, volume 1. Cambridge University Press, Cambridge, UK.

[455] Voit, E. O., Martens, H. A., and Omholt, S. W. (2015). 150 years of the mass action law. *PLOS Comput. Biol.*, 11(1):e1004012.

[456] Wang, E., Zhang, Q., Shen, B., Zhang, G., Lu, X., Wu, Q., and Wang, Y. (2014). Intel math kernel library. In *High-Performance Computing on the Intel® Xeon Phi™*, pages 167–188. Springer.

[457] Wang, R. S., Wu, L. Y., Li, Z. P., and Zhang, X. S. (2005). Haplotype reconstruction from SNP fragments by minimum error correction. *Bioinformatics*, 21(10):2456–2462.

[458] Wang, S. and Summers, R. M. (2012). Machine learning and radiology. *Med. Image Anal.*, 16(5):933–951.

[459] Wang, T. C., Taheri, J., and Zomaya, A. Y. (2012). Using genetic algorithm in reconstructing single individual haplotype with minimum error correction. *J. Biomed. Inform.*, 45(5):922–930.

[460] Wang, Y., Christley, S., Mjolsness, E., and Xie, X. (2010). Parameter inference for discretely observed stochastic kinetic models using stochastic gradient descent. *BMC Syst. Biol.*, 4(1):99.

[461] Wang, Y., Zhang, X.-S., and Chen, L. (2013). Computational systems biology in the big data era. *BMC Syst. Biol.*, 7(2):S1.

[462] Wang, Z. and Bovik, A. C. (2002). A universal image quality index. *IEEE Signal Process. Lett.*, 9(3):81–84.

[463] Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Proc.*, 13(4):600–612.

[464] Widder, S., Macía, J., and Solé, R. (2009). Monomeric bistability and the role of autoloops in gene regulation. *PLOS One*, 4(4):e5399.

[465] Wilhelm, T. (2009). The smallest chemical reaction system with bistability. *BMC Syst. Biol.*, 3(1):90.

[466] Wilkinson, D. (2009). Stochastic modelling for quantitative description of heterogeneous biological systems. *Nat. Rev. Genet.*, 10(2):122–133.

[467] Wilson, J. E. (2003). Isozymes of mammalian hexokinase: structure, subcellular localization and metabolic function. *J. Exp. Biol.*, 206(12):2049–2057.

[468] Wilt, N. (2013). *The CUDA Handbook: A Comprehensive Guide to GPU Programming*. Addison-Wesley, Boston, MA, USA.

[469] Wittig, U., Kania, R., Golebiewski, M., Rey, M., Shi, L., Jong, L., Algaa, E., Weidemann, A., Sauer-Danzwith, H., Mir, S., et al. (2011). SABIO-RK—database for biochemical reaction kinetics. *Nucleic Acids Res.*, 40(D1):D790–D796.

[470] Wolf, V., Goel, R., Mateescu, M., and Henzinger, T. A. (2010). Solving the chemical master equation using sliding windows. *BMC Syst. Biol.*, 4(1):42.

[471] Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.*, 1(1):67–82.

[472] Wuntch, T., Chen, R. F., and Vesell, E. S. (1970). Lactate dehydrogenase isozymes: kinetic properties at high enzyme concentrations. *Science*, 167(3914):63–65.

[473] Xiao, B., Tang, H., Jiang, Y., Li, W., and Wang, G. (2018). Brightness and contrast controllable image enhancement based on histogram specification. *Neurocomput.*, 275:2798–2809.

[474] Xie, J., Zhou, Y., and Chen, H. (2013). A novel bat algorithm based on differential operator and Lévy flights trajectory. *Comput. Intell. Neurosci.*, 2013.

[475] Xu, S. and Rahmat-Samii, Y. (2007). Boundary conditions in particle swarm optimization revisited. *IEEE Trans. Antennas Propag.*, 55(3):760–765.

[476] Xu, X., Xu, S., Jin, L., and Song, E. (2011). Characteristic analysis of Otsu threshold and its applications. *Pattern Recognit. Lett.*, 32(7):956–961.

[477] Xue, J. H. and Zhang, Y. J. (2012). Ridler and Calvard's, Kittler and Illingworth's and Otsu's methods for image thresholding. *Pattern Recogn. Lett.*, 33(6):793–797.

[478] Yadav, S. K., Mindur, J. E., Ito, K., and Dhib-Jalbut, S. (2015). Advances in the immunopathogenesis of multiple sclerosis. *Curr. Opin. Neurol.*, 28(3):206–219.

[479] Yang, B., Chen, Y., Zhao, Z., and Han, Q. (2006). A master-slave particle swarm optimization algorithm for solving constrained optimization problems. In *Proc. 6th IEEE World Congress Intelligent Control and Automation*, volume 1, pages 3208–3212. IEEE.

[480] Yang, J., Monine, M. I., Faeder, J. R., and Hlavacek, W. S. (2008). Kinetic monte carlo method for rule-based modeling of biochemical networks. *Phys. Rev. E. Stat. Nonlin. Soft. Matter. Phys.*, 78(3):031910.

[481] Yang, X. S. (2010). A new metaheuristic bat-inspired algorithm. In *Proc. Nature inspired cooperative strategies for optimization*, pages 65–74. Springer.

[482] Yankeelov, T. E., Mankoff, D. A., Schwartz, L. H., Lieberman, F. S., Buatti, J. M., Mountz, J. M., Erickson, B. J., Fennessy, F. M. M., Huang, W., Kalpathy-Cramer, J., et al. (2016). Quantitative imaging in cancer clinical trials. *Clin. Cancer Res.*, 22(2):284–290.

[483] Ye, Q. Z. and Danielsson, P. E. (1988). On minimum error thresholding and its implementations. *Pattern Recognit. Lett.*, 7(4):201–206.

[484] Yen, J. and Langari, R. (1999). *Fuzzy Logic: Intelligence, Control, and Information*. Prentice Hall, Upper Saddle River, NJ, USA.

[485] Yue, H., Brown, M., He, F., Jia, J., and Kell, D. B. (2008). Sensitivity analysis and robust experimental design of a signal transduction pathway system. *Int. J. Chem. Kinet.*, 40(11):730–741.

[486] Zaman, S., Lippman, S. I., Schneper, L., Slonim, N., and Broach, J. R. (2009). Glucose regulates transcription in yeast through a network of signaling pathways. *Mol. Syst. Biol.*, 5(1):245.

[487] Zhan, C., Situ, W., Yeung, L. F., Tsang, P. W. M., and Yang, G. (2014). A parameter estimation method for biological systems modelled by ODE/DDE models using spline approximation and differential evolution algorithm. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, 11(6):1066–1076.

[488] Zhang, J., Watson, L. T., and Cao, Y. (2010). A modified uniformization method for the solution of the chemical master equation. *Comput. Math. Appl.*, 59(1):573–584.

[489] Zhang, K., Calabrese, P., Nordborg, M., and Sun, F. (2002). Haplotype block structure and its applications to association studies: power and study designs. *Am. J. Hum. Genet.*, 71(6):1386–1394.

[490] Zhang, Y. J. (2001). A review of recent evaluation methods for image segmentation. In *Proc. 6th IEEE International Symposium on Signal Processing and its Applications*, volume 1, pages 148–151. IEEE.

[491] Zhou, Y., Liepe, J., Sheng, X., et al. (2011). GPU accelerated biochemical network simulation. *Bioinformatics*, 27(6):874–876.

[492] Zijdenbos, A. P., Dawant, B. M., Margolin, R. A., and Palmer, A. C. (1994). Morphometric analysis of white matter lesions in MR images: method and validation. *IEEE Trans. Med. Imaging*, 13(4):716–724.

[493] Zook, J. M., Chapman, B., Wang, J., Mittelman, D., Hofmann, O., Hide, W., and Salit, M. (2014). Integrating human sequence data sets provides a resource of benchmark SNP and indel genotype calls. *Nat. Biotech.*, 32(3):246–251.

[494] Zou, K. H., Warfield, S. K., Bharatha, A., Tempany, C. M. C., Kaus, M. R., Haker, S. J., Wells, W. M., Jolesz, F. A., and Kikinis, R. (2004). Statistical validation of image segmentation quality based on a spatial overlap index. *Acad. Radiol.*, 11(2):178–189.

Appendix A

Reaction-based models

In this appendix, the Reaction-Based Models (RBMs) used in the tests carried out in this thesis are described. In Section A.1 a theoretical model representing an autocatalytic, named Brusselator is described; Section A.2 presents the eukaryotic heat shock response model (HSR); the details about the human intracellular core metabolic pathways in a red blood cell are provided in Section A.3; Section A.4 is dedicated to the Prokaryotic auto-regulatory (PGN) gene network; the Ras/cAMP/PKA signaling pathway in *Saccharomyces cerevisiae* yeast is described in Section A.5; finally, Section A.6 explains the Schlögl model.

A.1 The Brusselator model

The Brusselator is a theoretical model representing an autocatalytic, oscillating chemical reaction [447, 22]. In an autocatlytic reaction a species acts to increase its producing reaction rate. Several autocatlytic systems have been studied to observe complex dynamics, including multiple steady-states and periodic orbits. Moreover, since the autocatalytic chemical reaction phenomenon plays a fundamental role in the breakdown of the stability of the thermodynamical branch, it gained ground during the years.

As shown in Table A.1, the Brusselator model is composed of 4 reactions involving 6 different species. The reactant A is converted in a final product E by means of the 4 reactions modifying 4 intermediate species (i.e., X, B, Y and D). Reactions R_2 and R_3 represent bimolecular and autocatalytic trimolecular reactions, respectively, while R_4 generates the product species E consuming the species X produced in R_1 from the species A. The initial molecular amounts of the species different from 0 are listed in Table A.2. Notice that the amounts of the species A and B are kept constant during the simulation.

Table A.1 List of the reactions of the Brusselator model.

No.	Reagents	Products	Constant
R_1	A	X	1
R_2	B + X	Y	1
R_3	2X + Y	3X	1
R_4	X	E	1

Table A.2 Initial molecular amounts of the Brusellator model.

Species	Initial amount
A	1
B	3
X	1
Y	1

A.2 The eukaryotic Heat Shock Response model

The Heat Shock Response (HSR) is a regulatory mechanism that allows the cell to quickly react to high temperatures and other forms of physiological and environmental stress [232]. In HSR, the heat shock protein (hsp) is fundamental to prevent mis-folding, and thus facilitating protein folding. In the absence of heat stress, the heat shock factor molecules (hsf) are present as monomers, usually bounded to hsp. On the contrary, in stress conditions, hsf forms first dimers and then trimers, which are the active components able to bind heat shock elements (hse) that activates genes encoding hsp. A sufficient concentration of hsp exerts a negative feedback by unbinding the complexes formed by a trimer of hsf and hse, thus inhibiting their synthesis. Finally, proteins mis-folding is formalized as a reaction whose rate depends on the system temperature; hsp binds to a mis-folded protein and helps its refolding.

In this thesis we considered the HSR model presented in [340], consisting of 17 reactions among 10 species (Table A.3). The initial concentrations of the species different from 0 are shown in Table A.4.

Table A.3 List of the reactions of the HSR model.

No.	Reagents	Products	Constant
R_1	2hsf	hsf_2	3.49
R_2	hsf_2	2hsf_2	0.19
R_3	hsf + hsf_2	hsf_3	1.07

R_4	hsf_3	$hsf + hsf_2$	$1 \cdot 10^{-9}$
R_5	$hsf_3 + hse$	hsf_3:hse	0.17
R_6	hsf_3:hse	$hsf_3 + hse$	$1.21 \cdot 10^{-6}$
R_7	hsf_3:hse	hsf_3:hse + hsp	$8.3 \cdot 10^{-3}$
R_8	hsp + hsf	hsp:hsf	9.74
R_9	hsp:hsf	hsp + hsf	3.56
R_{10}	$hsp + hsf_2$	hsp:hsf + hsf	2.33
R_{11}	$hsp + hsf_3$	hsp:hsf + 2hsf	$4.31 \cdot 10^{-5}$
R_{12}	hsp + hsf_3:hse	hsp:hsf + hse + 2hsf	$2.73 \cdot 10^{-7}$
R_{13}	hsp	$\emptyset$	$3.2 \cdot 10^{-5}$
R_{14}	prot	mfp	T dependent
R_{15}	hsp + mfp	hsp:mfp	$3.32 \cdot 10^{-3}$
R_{16}	hsp:mfp	hsp + mfp	4.44
R_{17}	hsp:mfp	hsp + prot	13.94

Table A.4 Initial concentrations of the HSR model.

Species	**Initial amount**
hsf	0.67
hsf_2	$8.7 \cdot 10^{-4}$
hsf_3	$1.2 \cdot 10^{-4}$
hse	29.73
hsf_3:hse	2.96
hsp	766.88
hsp:hsf	1403.13
mfp	517.352
hs mfp	71.65
prot	$1.15 \cdot 10^{8}$

The kinetic constant value of the reaction R_{14} depends by the temperature T of the environment, expressed in Celsius degrees. As shown in [340], its value can be calculated by modifying the formula proposed in [338] as follows:

$$\left(1 - \frac{0.4}{e^{T-37}}\right) \cdot 1.4^{T-37} \cdot 1.45 \cdot 10^{-5} \quad s^{-1}. \tag{A.1}$$

According to [252], this equation can be applied with T in the range $[37°C, 45°C]$.

A.3 A human intracellular core metabolic pathway in a red blood cell

In this thesis we consider the model of the red blood cell metabolism presented in [216]. The basic version of the model consists in 92 metabolites and 94 reactions describing the central pathways for carbohydrate metabolism, namely, glycolysis and pentose phosphate pathway, in a human red blood cell. Differently from [216], we do not consider the uptake of extracellular substrates. Instead, we consider the different isoforms of the enzyme hexokinase (HK), the first enzyme of the glycolythic pathway that converts glucose (GLC) into glucose-6-phosphate (G6P). By so doing, we extended the model obtaining 226 reactions (see Table A.5) among 114 species. The initial concentrations of the species along with $(50, 75, 100\%)$ knock-down interventions on the HK I isoform are shown in Table A.6

Table A.5 List of the reactions of the RBM of the human intracellular core metabolic pathways in a red blood cell. The estimated values of the kinetic parameters obtained by means of the PE presented in Section 5.4 are denoted by the asterisk “*”.

No.	Reagents	Products	Constant
R_1	hkE_0 + MgATP	hkEMgATP_0	$3.948 \cdot 10^{10}*$
R_2	hkE_1 + MgATP	hkEMgATP_1	$6.858 \cdot 10^{9}*$
R_3	hkE_2 + MgATP	hkEMgATP_2	$2.030 \cdot 10^{9}*$
R_4	hkEMgATP_0	hkE_0 + MgATP	$6.226 \cdot 10^{9}*$
R_5	hkEMgATP_1	hkE_1 + MgATP	$1.948 \cdot 10^{9}*$
R_6	hkEMgATP_2	hkE_2 + MgATP	$1.418 \cdot 10^{9}*$
R_7	hkEMgATP_0 + GLC	hkEMgATPGLC_0	$1.925 \cdot 10^{11}*$
R_8	hkEMgATP_1 + GLC	hkEMgATPGLC_1	$3.066 \cdot 10^{8}*$
R_9	hkEMgATP_2 + GLC	hkEMgATPGLC_2	$3.209 \cdot 10^{10}*$
R_{10}	hkEMgATPGLC_0	hkEMgATP_0 + GLC	$1.021 \cdot 10^{10}*$
R_{11}	hkEMgATPGLC_1	hkEMgATP_1 + GLC	$2.599 \cdot 10^{10}*$
R_{12}	hkEMgATPGLC_2	hkEMgATP_2 + GLC	$3.995 \cdot 10^{9}*$
R_{13}	hkE_0 + GLC	hkEGLC_0	$6.943 \cdot 10^{10}*$
R_{14}	hkE_1 + GLC	hkEGLC_1	$4.465 \cdot 10^{10}*$
R_{15}	hkE_2 + GLC	hkEGLC_2	$1.358 \cdot 10^{10}*$
R_{16}	hkEGLC_0	hkE_0 + GLC	$3.498 \cdot 10^{10}*$
R_{17}	hkEGLC_1	hkE_1 + GLC	$1.157 \cdot 10^{10}*$
R_{18}	hkEGLC_2	hkE_2 + GLC	$1.180 \cdot 10^{8}*$
R_{19}	MgATP + hkEGLC_0	hkEMgATPGLC_0	$1.048 \cdot 10^{10}*$
R_{20}	MgATP + hkEGLC_1	hkEMgATPGLC_1	$3.375 \cdot 10^{10}*$
R_{21}	MgATP + hkEGLC_2	hkEMgATPGLC_2	$9.843 \cdot 10^{9}*$

R_{22}	hkEMgATPGLC_0	MgATP + hkEGLC_0	$3.307 \cdot 10^{9}$*
R_{23}	hkEMgATPGLC_1	MgATP + hkEGLC_1	$2.257 \cdot 10^{10}$*
R_{24}	hkEMgATPGLC_2	MgATP + hkEGLC_2	$1.289 \cdot 10^{10}$*
R_{25}	hkEMgATPGLC_0	hkEMgADPG6P_0	$8.281 \cdot 10^{4}$*
R_{26}	hkEMgATPGLC_1	hkEMgADPG6P_1	$4.610 \cdot 10^{6}$*
R_{27}	hkEMgATPGLC_2	hkEMgADPG6P_2	$6.869 \cdot 10^{5}$*
R_{28}	hkEMgADPG6P_0	hkEMgATPGLC_0	$1.505 \cdot 10^{5}$*
R_{29}	hkEMgADPG6P_1	hkEMgATPGLC_1	$5.613 \cdot 10^{4}$*
R_{30}	hkEMgADPG6P_2	hkEMgATPGLC_2	$1.399 \cdot 10^{5}$*
R_{31}	hkEMgADPG6P_0	hkEG6P_0 + MgADP	$6.013 \cdot 10^{8}$*
R_{32}	hkEMgADPG6P_1	hkEG6P_1 + MgADP	$9.520 \cdot 10^{8}$*
R_{33}	hkEMgADPG6P_2	hkEG6P_2 + MgADP	$4.419 \cdot 10^{9}$*
R_{34}	hkEG6P_0 + MgADP	hkEMgADPG6P_0	$1.258 \cdot 10^{10}$*
R_{35}	hkEG6P_1 + MgADP	hkEMgADPG6P_1	$1.196 \cdot 10^{10}$*
R_{36}	hkEG6P_2 + MgADP	hkEMgADPG6P_2	$5.685 \cdot 10^{9}$*
R_{37}	hkEG6P_0	hkE_0 + G6P	$1.232 \cdot 10^{10}$*
R_{38}	hkEG6P_1	hkE_1 + G6P	$1.518 \cdot 10^{8}$*
R_{39}	hkEG6P_2	hkE_2 + G6P	$7.616 \cdot 10^{8}$*
R_{40}	hkE_0 + G6P	hkEG6P_0	$1.896 \cdot 10^{10}$*
R_{41}	hkE_1 + G6P	hkEG6P_1	$7.445 \cdot 10^{10}$*
R_{42}	hkE_2 + G6P	hkEG6P_2	$7.039 \cdot 10^{10}$*
R_{43}	hkEMgADPG6P_0	G6P + hkEMgADP_0	$1.163 \cdot 10^{9}$*
R_{44}	hkEMgADPG6P_1	G6P + hkEMgADP_1	$2.004 \cdot 10^{9}$*
R_{45}	hkEMgADPG6P_2	G6P + hkEMgADP_2	$1.019 \cdot 10^{10}$*
R_{46}	G6P + hkEMgADP_0	hkEMgADPG6P_0	$1.004 \cdot 10^{11}$*
R_{47}	G6P + hkEMgADP_1	hkEMgADPG6P_1	$2.605 \cdot 10^{10}$*
R_{48}	G6P + hkEMgADP_2	hkEMgADPG6P_2	$7.624 \cdot 10^{9}$*
R_{49}	hkEMgADP_0	hkE_0 + MgADP	$1.225 \cdot 10^{8}$*
R_{50}	hkEMgADP_1	hkE_1 + MgADP	$3.246 \cdot 10^{9}$*
R_{51}	hkEMgADP_2	hkE_2 + MgADP	$1.920 \cdot 10^{9}$*
R_{52}	hkE_0 + MgADP	hkEMgADP_0	$1.311 \cdot 10^{9}$*
R_{53}	hkE_1 + MgADP	hkEMgADP_1	$7.685 \cdot 10^{9}$*
R_{54}	hkE_2 + MgADP	hkEMgADP_2	$2.757 \cdot 10^{10}$*
R_{55}	hkEGLC_0 + GSH	hkEGLCGSH_0	$7.605 \cdot 10^{8}$*
R_{56}	hkEGLC_1 + GSH	hkEGLCGSH_1	$3.157 \cdot 10^{8}$*
R_{57}	hkEGLC_2 + GSH	hkEGLCGSH_2	$3.176 \cdot 10^{8}$*
R_{58}	hkEGLCGSH_0	hkEGLC_0 + GSH	$5.035 \cdot 10^{8}$*
R_{59}	hkEGLCGSH_1	hkEGLC_1 + GSH	$1.021 \cdot 10^{9}$*
R_{60}	hkEGLCGSH_2	hkEGLC_2 + GSH	$2.304 \cdot 10^{10}$*

R_{61}	hkEGLC_0 + DPG23	hkEGLCDPG23_0	$1.770 \cdot 10^9$*
R_{62}	hkEGLC_1 + DPG23	hkEGLCDPG23_1	$9.980 \cdot 10^8$*
R_{63}	hkEGLC_2 + DPG23	hkEGLCDPG23_2	$6.864 \cdot 10^8$*
R_{64}	hkEGLCDPG23_0	hkEGLC_0 + DPG23	$7.162 \cdot 10^9$*
R_{65}	hkEGLCDPG23_1	hkEGLC_1 + DPG23	$1.121 \cdot 10^{10}$*
R_{66}	hkEGLCDPG23_2	hkEGLC_2 + DPG23	$4.799 \cdot 10^9$*
R_{67}	hkE_0 + Phosi	hkEPhosi_0	$6.966 \cdot 10^{10}$*
R_{68}	hkE_1 + Phosi	hkEPhosi_1	$3.806 \cdot 10^9$*
R_{69}	hkE_2 + Phosi	hkEPhosi_2	$3.734 \cdot 10^{10}$*
R_{70}	hkEPhosi_0	hkE_0 + Phosi	$1.724 \cdot 10^9$*
R_{71}	hkEPhosi_1	hkE_1 + Phosi	$1.464 \cdot 10^8$*
R_{72}	hkEPhosi_2	hkE_2 + Phosi	$4.894 \cdot 10^7$*
R_{73}	hkEGLC_0 + G6P	hkEGLCG6P_0	$8.938 \cdot 10^{10}$*
R_{74}	hkEGLC_1 + G6P	hkEGLCG6P_1	$2.703 \cdot 10^{11}$*
R_{75}	hkEGLC_2 + G6P	hkEGLCG6P_2	$1.321 \cdot 10^{11}$*
R_{76}	hkEGLCG6P_0	hkEGLC_0 + G6P	$1.254 \cdot 10^8$*
R_{77}	hkEGLCG6P_1	hkEGLC_1 + G6P	$1.028 \cdot 10^9$*
R_{78}	hkEGLCG6P_2	hkEGLC_2 + G6P	$8.807 \cdot 10^8$*
R_{79}	G6P	F6P	$1.151 \cdot 10^3$
R_{80}	F6P	G6P	$2.676 \cdot 10^3$
R_{81}	MgATP + pfkER	pfkERMgATP	$1.000 \cdot 10^9$
R_{82}	pfkERMgATP	MgATP + pfkER	$1.000 \cdot 10^9$
R_{83}	F6P + pfkERMgATP	pfkERMgATPF6P	$1.000 \cdot 10^9$
R_{84}	pfkERMgATPF6P	F6P + pfkERMgATP	$1.000 \cdot 10^9$
R_{85}	pfkERMgATPF6P	pfkERMgADPFDP	$1.000 \cdot 10^9$
R_{86}	pfkERMgADPFDP	pfkERMgATPF6P	$8.475 \cdot 10^4$
R_{87}	pfkERMgADPFDP	FDP + pfkERMgADP	$1.000 \cdot 10^9$
R_{88}	FDP + pfkERMgADP	pfkERMgADPFDP	$1.000 \cdot 10^9$
R_{89}	pfkERMgADP	MgADP + pfkER	$1.000 \cdot 10^9$
R_{90}	MgADP + pfkER	pfkERMgADP	$1.000 \cdot 10^9$
R_{91}	pfkER + AMP	pfkERAMP	$1.000 \cdot 10^9$
R_{92}	pfkERAMP	pfkER + AMP	$1.000 \cdot 10^9$
R_{93}	AMP + pfkERAMP	pfkERAMP2	$1.000 \cdot 10^9$
R_{94}	pfkERAMP2	AMP + pfkERAMP	$1.000 \cdot 10^9$
R_{95}	AMP + pfkERAMP2	pfkERAMP3	$1.000 \cdot 10^9$
R_{96}	pfkERAMP3	AMP + pfkERAMP2	$1.000 \cdot 10^9$
R_{97}	AMP + pfkERAMP3	pfkERAMP4	$1.000 \cdot 10^9$
R_{98}	pfkERAMP4	AMP + pfkERAMP3	$1.000 \cdot 10^9$
R_{99}	F6P + pfkER	pfkERF6P	$1.000 \cdot 10^9$

R_{100}	pfkERF6P	F6P + pfkER	$1.000 \cdot 10^9$
R_{101}	F6P + pfkERF6P	pfkERF6P2	$1.000 \cdot 10^9$
R_{102}	pfkERF6P2	F6P + pfkERF6P	$1.000 \cdot 10^9$
R_{103}	F6P + pfkERF6P2	pfkERF6P3	$1.000 \cdot 10^9$
R_{104}	pfkERF6P3	F6P + pfkERF6P2	$1.000 \cdot 10^9$
R_{105}	F6P + pfkERF6P3	pfkERF6P4	$1.000 \cdot 10^9$
R_{106}	pfkERF6P4	F6P + pfkERF6P3	$1.000 \cdot 10^9$
R_{107}	ATP + pfkET	pfkETATP	$1.000 \cdot 10^9$
R_{108}	pfkETATP	ATP + pfkET	$1.000 \cdot 10^9$
R_{109}	ATP + pfkETATP	pfkETATP2	$1.000 \cdot 10^9$
R_{110}	pfkETATP2	ATP + pfkETATP	$1.000 \cdot 10^9$
R_{111}	ATP + pfkETATP2	pfkETATP3	$1.000 \cdot 10^9$
R_{112}	pfkETATP3	ATP + pfkETATP2	$1.000 \cdot 10^9$
R_{113}	ATP + pfkETATP3	pfkETATP4	$1.000 \cdot 10^9$
R_{114}	pfkETATP4	ATP + pfkETATP3	$1.000 \cdot 10^9$
R_{115}	pfkET + Mg	pfkETMg	$1.000 \cdot 10^9$
R_{116}	pfkETMg	pfkET + Mg	$1.000 \cdot 10^9$
R_{117}	Mg + pfkETMg	pfkETMg2	$1.000 \cdot 10^9$
R_{118}	pfkETMg2	Mg + pfkETMg	$1.000 \cdot 10^9$
R_{119}	Mg + pfkETMg2	pfkETMg3	$1.000 \cdot 10^9$
R_{120}	pfkETMg3	Mg + pfkETMg2	$1.000 \cdot 10^9$
R_{121}	Mg + pfkETMg3	pfkETMg4	$1.000 \cdot 10^9$
R_{122}	pfkETMg4	Mg + pfkETMg3	$1.000 \cdot 10^9$
R_{123}	FDP	DHAP + GAP	$1.457 \cdot 10^2$
R_{124}	DHAP + GAP	FDP	1.181
R_{125}	DHAP	GAP	7.926
R_{126}	GAP	DHAP	$4.529 \cdot 10^{-1}$
R_{127}	Phosi + GAP + NAD	DPG13 + NADH	$1.425 \cdot 10^5$
R_{128}	DPG13 + NADH	Phosi + GAP + NAD	$5.280 \cdot 10^6$
R_{129}	DPG13 + ADP	ATP + PG3	$2.606 \cdot 10^4$
R_{130}	ATP + PG3	DPG13 + ADP	$1.448 \cdot 10^1$
R_{131}	DPG13 + dpgmE	dpgmEDPG13	$2.880 \cdot 10^8$
R_{132}	dpgmEDPG13	DPG13 + dpgmE	$1.510 \cdot 10^6$
R_{133}	dpgmEDPG13	dpgmEDPG23	$1.000 \cdot 10^8$
R_{134}	dpgmEDPG23	dpgmEDPG13	$1.000 \cdot 10^5$
R_{135}	dpgmEDPG23	DPG23 + dpgmE	$6.480 \cdot 10^6$
R_{136}	DPG23 + dpgmE	dpgmEDPG23	$8.800 \cdot 10^7$
R_{137}	dpgmEDPG23	dpgmEPhosiDPG3	$1.980 \cdot 10^3$
R_{138}	dpgmEPhosiDPG3	dpgmEDPG23	$3.600 \cdot 10^7$

R_{139}	dpgmEPhosiDPG3	PG3 + dpgmEPhosi	$3.600 \cdot 10^{11}$
R_{140}	PG3 + dpgmEPhosi	dpgmEPhosiDPG3	$6.660 \cdot 10^{8}$
R_{141}	dpgmEPhosi	Phosi + dpgmE	$6.840 \cdot 10^{2}$
R_{142}	Phosi + dpgmE	dpgmEPhosi	$1.000 \cdot 10^{-20}$
R_{143}	PG3	PG2	$5.383 \cdot 10^{1}$
R_{144}	PG2	PG3	7.916
R_{145}	PG2	PEP	$5.822 \cdot 10^{2}$
R_{146}	PEP	PG2	$3.435 \cdot 10^{2}$
R_{147}	ADP + PEP	ATP + PYR	$5.169 \cdot 10^{2}$
R_{148}	ATP + PYR	ADP + PEP	$5.169 \cdot 10^{-1}$
R_{149}	NADH + PYR	NAD + LAC	$1.045 \cdot 10^{3}$
R_{150}	NAD + LAC	NADH + PYR	2.340
R_{151}	AMP	Phosi + ADO	3.278
R_{152}	Phosi + ADO	AMP	3.278
R_{153}	ADO	INO	$7.450 \cdot 10^{2}$
R_{154}	INO	ADO	$7.450 \cdot 10^{2}$
R_{155}	ADO + akE	akEADO	$1.000 \cdot 10^{7}$
R_{156}	akEADO	ADO + akE	$1.000 \cdot 10^{5}$
R_{157}	MgATP + akEADO	akEADOMgATP	$2.000 \cdot 10^{5}$
R_{158}	akEADOMgATP	MgATP + akEADO	$1.000 \cdot 10^{5}$
R_{159}	akEADOMgATP	akEAMPMgADP	$5.000 \cdot 10^{7}$
R_{160}	akEAMPMgADP	akEADOMgATP	$1.000 \cdot 10^{5}$
R_{161}	akEAMPMgADP	MgADP + akEAMP	$2.493 \cdot 10^{3}$
R_{162}	MgADP + akEAMP	akEAMPMgADP	$1.000 \cdot 10^{5}$
R_{163}	akEAMP	AMP + akE	$5.230 \cdot 10^{6}$
R_{164}	AMP + akE	akEAMP	$1.000 \cdot 10^{5}$
R_{165}	AMP + akEADO	akEAMPADO	$6.090 \cdot 10^{4}$
R_{166}	akEAMPADO	AMP + akEADO	$1.000 \cdot 10^{5}$
R_{167}	ADO + akET	akETADO	$1.841 \cdot 10^{4}$
R_{168}	akETADO	ADO + akET	$1.000 \cdot 10^{5}$
R_{169}	akETADO	akEADO	$2.050 \cdot 10^{8}$
R_{170}	akEADO	akETADO	$1.000 \cdot 10^{5}$
R_{171}	akET	akE	$4.000 \cdot 10^{5}$
R_{172}	akE	akET	$1.000 \cdot 10^{5}$
R_{173}	2ADP	AMP + ATP	$7.820 \cdot 10^{1}$
R_{174}	AMP + ATP	ADP	$4.739 \cdot 10^{1}$
R_{175}	AMP	IMP	6.174
R_{176}	IMP	AMP	$6.174 \cdot 10^{1}$
R_{177}	ATP	Phosi + ADP	$9.742 \cdot 10^{-1}$

R_{178}	Phosi + ADP	ATP	$9.740 \cdot 10^{-4}$
R_{179}	PRPP + ADE	Phosi + AMP	$3.481 \cdot 10^{3}$
R_{180}	2Phosi + AMP	PRPP + ADE	$3.481 \cdot 10^{-2}$
R_{181}	NADP + g6pdER	g6pdERNADP	$3.960 \cdot 10^{8}$
R_{182}	g6pdERNADP	NADP + g6pdER	$3.130 \cdot 10^{6}$
R_{183}	G6P + g6pdERNADP	g6pdERNADPG6P	$9.360 \cdot 10^{7}$
R_{184}	g6pdERNADPG6P	G6P + g6pdERNADP	$1.080 \cdot 10^{6}$
R_{185}	g6pdERNADPG6P	g6pdERNADPHGL6P	$2.700 \cdot 10^{6}$
R_{186}	g6pdERNADPHGL6P	g6pdERNADPG6P	$7.200 \cdot 10^{6}$
R_{187}	g6pdERNADPHGL6P	GL6P + g6pdERNADPH	$3.960 \cdot 10^{12}$
R_{188}	GL6P + g6pdERNADPH	g6pdERNADPHGL6P	$7.920 \cdot 10^{5}$
R_{189}	g6pdERNADPH	g6pdER + NADPH	$1.400 \cdot 10^{12}$
R_{190}	g6pdER + NADPH	g6pdERNADPH	$3.600 \cdot 10^{4}$
R_{191}	GL6P	GO6P	$1.223 \cdot 10^{2}$
R_{192}	GO6P	GL6P	$1.223 \cdot 10^{-1}$
R_{193}	NADP + GO6P	NADPH + RU5P	$2.929 \cdot 10^{4}$
R_{194}	NADPH + RU5P	NADP + GO6P	$2.929 \cdot 10^{1}$
R_{195}	NADPH	GSH + NADP	6.385
R_{196}	2GSH + NADP	NADPH	$6.385 \cdot 10^{-2}$
R_{197}	GSH		$3.111 \cdot 10^{-1}$
R_{198}		GSH	$1.556 \cdot 10^{-1}$
R_{199}	RU5P	R5P	$1.854 \cdot 10^{3}$
R_{200}	R5P	RU5P	$7.212 \cdot 10^{2}$
R_{201}	RU5P	X5P	$2.253 \cdot 10^{4}$
R_{202}	X5P	RU5P	$7.510 \cdot 10^{3}$
R_{203}	R5P + X5P	GAP + S7P	$1.842 \cdot 10^{3}$
R_{204}	GAP + S7P	R5P + X5P	$1.535 \cdot 10^{3}$
R_{205}	X5P + E4P	F6P + GAP	$1.521 \cdot 10^{3}$
R_{206}	F6P + GAP	X5P + E4P	$1.477 \cdot 10^{2}$
R_{207}	GAP + S7P	F6P + E4P	$1.179 \cdot 10^{3}$
R_{208}	F6P + E4P	GAP + S7P	$1.123 \cdot 10^{3}$
R_{209}	IMP	Phosi + INO	$6.661 \cdot 10^{-2}$
R_{210}	Phosi + INO	IMP	$6.700 \cdot 10^{-5}$
R_{211}	Phosi + INO	HX + R1P	$2.044 \cdot 10^{3}$
R_{212}	HX + R1P	Phosi + INO	$1.858 \cdot 10^{4}$
R_{213}	R1P	R5P	3.777
R_{214}	R5P	R1P	$2.840 \cdot 10^{-1}$
R_{215}	ATP + R5P	AMP + PRPP	7.081
R_{216}	AMP + PRPP	ATP + R5P	$1.170 \cdot 10^{-2}$

R_{217}	PRPP + HX	Phosi + IMP	$1.450 \cdot 10^{4}$
R_{218}	2Phosi + IMP	PRPP + HX	1.450
R_{219}	ATP + Mg	MgATP	$1.000 \cdot 10^{8}$
R_{220}	MgATP	ATP + Mg	$1.230 \cdot 10^{9}$
R_{221}	Mg + ADP	MgADP	$1.000 \cdot 10^{8}$
R_{222}	MgADP	Mg + ADP	$1.230 \cdot 10^{8}$
R_{223}	AMP + Mg	MgAMP	$1.000 \cdot 10^{8}$
R_{224}	MgAMP	AMP + Mg	$4.500 \cdot 10^{6}$
R_{225}	DPG23 + Mg	MgDPG23	$1.000 \cdot 10^{8}$
R_{226}	MgDPG23	DPG23 + Mg	$5.990 \cdot 10^{7}$

Table A.6 Initial concentrations of the species of the model of human intracellular core metabolic pathways in a red blood cell.

Species	**Baseline**	**Knock-down 50**	**Knock-down 75**	**Knock-down 100**
hkE_0	$5.150 \cdot 10^{-8}$	$5.150 \cdot 10^{-8}$	$5.150 \cdot 10^{-8}$	$5.150 \cdot 10^{-8}$
hkE_1	$1.545 \cdot 10^{-7}$	$1.545 \cdot 10^{-7}$	$1.545 \cdot 10^{-7}$	$1.545 \cdot 10^{-7}$
hkE_2	$3.090 \cdot 10^{-7}$	$1.545 \cdot 10^{-7}$	$7.725 \cdot 10^{-8}$	0.000
MgATP	$3.167 \cdot 10^{-2}$	$3.167 \cdot 10^{-2}$	$3.167 \cdot 10^{-2}$	$3.167 \cdot 10^{-2}$
hkEMgATP_0	$1.620 \cdot 10^{-9}$	$1.620 \cdot 10^{-9}$	$1.620 \cdot 10^{-9}$	$1.620 \cdot 10^{-9}$
hkEMgATP_1	$4.860 \cdot 10^{-9}$	$4.860 \cdot 10^{-9}$	$4.860 \cdot 10^{-9}$	$4.860 \cdot 10^{-9}$
hkEMgATP_2	$9.720 \cdot 10^{-9}$	$4.860 \cdot 10^{-9}$	$2.430 \cdot 10^{-9}$	0.000
GLC	5.000	5.000	5.000	5.000
hkEMgATPGLC_0	$1.730 \cdot 10^{-7}$	$1.730 \cdot 10^{-7}$	$1.730 \cdot 10^{-7}$	$1.730 \cdot 10^{-7}$
hkEMgATPGLC_1	$5.190 \cdot 10^{-7}$	$5.190 \cdot 10^{-7}$	$5.190 \cdot 10^{-7}$	$5.190 \cdot 10^{-7}$
hkEMgATPGLC_2	$1.038 \cdot 10^{-6}$	$5.190 \cdot 10^{-7}$	$2.595 \cdot 10^{-7}$	0.000
hkEGLC_0	$5.490 \cdot 10^{-6}$	$5.490 \cdot 10^{-6}$	$5.490 \cdot 10^{-6}$	$5.490 \cdot 10^{-6}$
hkEGLC_1	$1.647 \cdot 10^{-5}$	$1.647 \cdot 10^{-5}$	$1.647 \cdot 10^{-5}$	$1.647 \cdot 10^{-5}$
hkEGLC_2	$3.294 \cdot 10^{-5}$	$1.647 \cdot 10^{-5}$	$8.235 \cdot 10^{-6}$	0.000
hkEMgADPG6P_0	$3.010 \cdot 10^{-9}$	$3.010 \cdot 10^{-9}$	$3.010 \cdot 10^{-9}$	$3.010 \cdot 10^{-9}$
hkEMgADPG6P_1	$9.030 \cdot 10^{-9}$	$9.030 \cdot 10^{-9}$	$9.030 \cdot 10^{-9}$	$9.030 \cdot 10^{-9}$
hkEMgADPG6P_2	$1.806 \cdot 10^{-8}$	$9.030 \cdot 10^{-9}$	$4.515 \cdot 10^{-9}$	0.000
hkEG6P_0	$4.240 \cdot 10^{-8}$	$4.240 \cdot 10^{-8}$	$4.240 \cdot 10^{-8}$	$4.240 \cdot 10^{-8}$
hkEG6P_1	$1.272 \cdot 10^{-7}$	$1.272 \cdot 10^{-7}$	$1.272 \cdot 10^{-7}$	$1.272 \cdot 10^{-7}$
hkEG6P_2	$2.544 \cdot 10^{-7}$	$1.272 \cdot 10^{-7}$	$6.360 \cdot 10^{-8}$	0.000
MgADP	$5.553 \cdot 10^{-2}$	$5.553 \cdot 10^{-2}$	$5.553 \cdot 10^{-2}$	$5.553 \cdot 10^{-2}$
G6P	$3.800 \cdot 10^{-2}$	$3.800 \cdot 10^{-2}$	$3.800 \cdot 10^{-2}$	$3.800 \cdot 10^{-2}$
hkEMgADP_0	$3.240 \cdot 10^{-9}$	$3.240 \cdot 10^{-9}$	$3.240 \cdot 10^{-9}$	$3.240 \cdot 10^{-9}$
hkEMgADP_1	$9.720 \cdot 10^{-9}$	$9.720 \cdot 10^{-9}$	$9.720 \cdot 10^{-9}$	$9.720 \cdot 10^{-9}$

hkEMgADP_2	$1.944 \cdot 10^{-8}$	$9.720 \cdot 10^{-9}$	$4.860 \cdot 10^{-9}$	0.000
GSH	3.200	3.200	3.200	3.200
hkEGLCGSH_0	$5.860 \cdot 10^{-6}$	$5.860 \cdot 10^{-6}$	$5.860 \cdot 10^{-6}$	$5.860 \cdot 10^{-6}$
hkEGLCGSH_1	$1.758 \cdot 10^{-5}$	$1.758 \cdot 10^{-5}$	$1.758 \cdot 10^{-5}$	$1.758 \cdot 10^{-5}$
hkEGLCGSH_2	$3.516 \cdot 10^{-5}$	$1.758 \cdot 10^{-5}$	$8.790 \cdot 10^{-6}$	0.000
DPG23	4.500	4.500	4.500	4.500
hkEGLCDPG23_0	$6.180 \cdot 10^{-6}$	$6.180 \cdot 10^{-6}$	$6.180 \cdot 10^{-6}$	$6.180 \cdot 10^{-6}$
hkEGLCDPG23_1	$1.854 \cdot 10^{-5}$	$1.854 \cdot 10^{-5}$	$1.854 \cdot 10^{-5}$	$1.854 \cdot 10^{-5}$
hkEGLCDPG23_2	$3.708 \cdot 10^{-5}$	$1.854 \cdot 10^{-5}$	$9.270 \cdot 10^{-6}$	0.000
Phosi	1.200	1.200	1.200	1.200
hkEPhosi_0	$4.390 \cdot 10^{-6}$	$4.390 \cdot 10^{-6}$	$4.390 \cdot 10^{-6}$	$4.390 \cdot 10^{-6}$
hkEPhosi_1	$1.317 \cdot 10^{-5}$	$1.317 \cdot 10^{-5}$	$1.317 \cdot 10^{-5}$	$1.317 \cdot 10^{-5}$
hkEPhosi_2	$2.634 \cdot 10^{-5}$	$1.317 \cdot 10^{-5}$	$6.585 \cdot 10^{-6}$	0.000
hkEGLCG6P_0	$6.260 \cdot 10^{-6}$	$6.260 \cdot 10^{-6}$	$6.260 \cdot 10^{-6}$	$6.260 \cdot 10^{-6}$
hkEGLCG6P_1	$1.878 \cdot 10^{-5}$	$1.878 \cdot 10^{-5}$	$1.878 \cdot 10^{-5}$	$1.878 \cdot 10^{-5}$
hkEGLCG6P_2	$3.756 \cdot 10^{-5}$	$1.878 \cdot 10^{-5}$	$9.390 \cdot 10^{-6}$	0.000
F6P	$1.600 \cdot 10^{-2}$	$1.600 \cdot 10^{-2}$	$1.600 \cdot 10^{-2}$	$1.600 \cdot 10^{-2}$
pfkER	$4.400 \cdot 10^{-6}$	$4.400 \cdot 10^{-6}$	$4.400 \cdot 10^{-6}$	$4.400 \cdot 10^{-6}$
pfkERMgATP	$1.380 \cdot 10^{-7}$	$1.380 \cdot 10^{-7}$	$1.380 \cdot 10^{-7}$	$1.380 \cdot 10^{-7}$
pfkERMgATPF6P	$1.110 \cdot 10^{-9}$	$1.110 \cdot 10^{-9}$	$1.110 \cdot 10^{-9}$	$1.110 \cdot 10^{-9}$
pfkERMgADPFDP	$2.970 \cdot 10^{-9}$	$2.970 \cdot 10^{-9}$	$2.970 \cdot 10^{-9}$	$2.970 \cdot 10^{-9}$
FDP	$7.600 \cdot 10^{-3}$	$7.600 \cdot 10^{-3}$	$7.600 \cdot 10^{-3}$	$7.600 \cdot 10^{-3}$
pfkERMgADP	$2.460 \cdot 10^{-7}$	$2.460 \cdot 10^{-7}$	$2.460 \cdot 10^{-7}$	$2.460 \cdot 10^{-7}$
AMP	$8.000 \cdot 10^{-2}$	$8.000 \cdot 10^{-2}$	$8.000 \cdot 10^{-2}$	$8.000 \cdot 10^{-2}$
pfkERAMP	$3.520 \cdot 10^{-7}$	$3.520 \cdot 10^{-7}$	$3.520 \cdot 10^{-7}$	$3.520 \cdot 10^{-7}$
pfkERAMP2	$2.820 \cdot 10^{-8}$	$2.820 \cdot 10^{-8}$	$2.820 \cdot 10^{-8}$	$2.820 \cdot 10^{-8}$
pfkERAMP3	$2.250 \cdot 10^{-9}$	$2.250 \cdot 10^{-9}$	$2.250 \cdot 10^{-9}$	$2.250 \cdot 10^{-9}$
pfkERAMP4	$1.800 \cdot 10^{-10}$	$1.800 \cdot 10^{-10}$	$1.800 \cdot 10^{-10}$	$1.800 \cdot 10^{-10}$
pfkERF6P	$7.040 \cdot 10^{-8}$	$7.040 \cdot 10^{-8}$	$7.040 \cdot 10^{-8}$	$7.040 \cdot 10^{-8}$
pfkERF6P2	$1.130 \cdot 10^{-9}$	$1.130 \cdot 10^{-9}$	$1.130 \cdot 10^{-9}$	$1.130 \cdot 10^{-9}$
pfkERF6P3	$1.800 \cdot 10^{-11}$	$1.800 \cdot 10^{-11}$	$1.800 \cdot 10^{-11}$	$1.800 \cdot 10^{-11}$
pfkERF6P4	$2.890 \cdot 10^{-13}$	$2.890 \cdot 10^{-13}$	$2.890 \cdot 10^{-13}$	$2.890 \cdot 10^{-13}$
ATP	1.540	1.540	1.540	1.540
pfkET	$3.080 \cdot 10^{-6}$	$3.080 \cdot 10^{-6}$	$3.080 \cdot 10^{-6}$	$3.080 \cdot 10^{-6}$
pfkETATP	$4.740 \cdot 10^{-6}$	$4.740 \cdot 10^{-6}$	$4.740 \cdot 10^{-6}$	$4.740 \cdot 10^{-6}$
pfkETATP2	$7.310 \cdot 10^{-6}$	$7.310 \cdot 10^{-6}$	$7.310 \cdot 10^{-6}$	$7.310 \cdot 10^{-6}$
pfkETATP3	$1.130 \cdot 10^{-5}$	$1.130 \cdot 10^{-5}$	$1.130 \cdot 10^{-5}$	$1.130 \cdot 10^{-5}$
pfkETATP4	$1.730 \cdot 10^{-5}$	$1.730 \cdot 10^{-5}$	$1.730 \cdot 10^{-5}$	$1.730 \cdot 10^{-5}$
Mg	$2.539 \cdot 10^{-1}$	$2.539 \cdot 10^{-1}$	$2.539 \cdot 10^{-1}$	$2.539 \cdot 10^{-1}$

pfkETMg	$7.820\cdot 10^{-7}$	$7.820\cdot 10^{-7}$	$7.820\cdot 10^{-7}$	$7.820\cdot 10^{-7}$
pfkETMg2	$1.990\cdot 10^{-7}$	$1.990\cdot 10^{-7}$	$1.990\cdot 10^{-7}$	$1.990\cdot 10^{-7}$
pfkETMg3	$5.040\cdot 10^{-8}$	$5.040\cdot 10^{-8}$	$5.040\cdot 10^{-8}$	$5.040\cdot 10^{-8}$
pfkETMg4	$1.280\cdot 10^{-8}$	$1.280\cdot 10^{-8}$	$1.280\cdot 10^{-8}$	$1.280\cdot 10^{-8}$
DHAP	$1.400\cdot 10^{-1}$	$1.400\cdot 10^{-1}$	$1.400\cdot 10^{-1}$	$1.400\cdot 10^{-1}$
GAP	$6.700\cdot 10^{-3}$	$6.700\cdot 10^{-3}$	$6.700\cdot 10^{-3}$	$6.700\cdot 10^{-3}$
NAD	$5.840\cdot 10^{-2}$	$5.840\cdot 10^{-2}$	$5.840\cdot 10^{-2}$	$5.840\cdot 10^{-2}$
DPG13	$4.000\cdot 10^{-4}$	$4.000\cdot 10^{-4}$	$4.000\cdot 10^{-4}$	$4.000\cdot 10^{-4}$
NADH	$3.060\cdot 10^{-2}$	$3.060\cdot 10^{-2}$	$3.060\cdot 10^{-2}$	$3.060\cdot 10^{-2}$
ADP	$2.700\cdot 10^{-1}$	$2.700\cdot 10^{-1}$	$2.700\cdot 10^{-1}$	$2.700\cdot 10^{-1}$
PG3	$4.500\cdot 10^{-2}$	$4.500\cdot 10^{-2}$	$4.500\cdot 10^{-2}$	$4.500\cdot 10^{-2}$
dpgmE	$2.220\cdot 10^{-5}$	$2.220\cdot 10^{-5}$	$2.220\cdot 10^{-5}$	$2.220\cdot 10^{-5}$
dpgmEDPG13	$1.360\cdot 10^{-6}$	$1.360\cdot 10^{-6}$	$1.360\cdot 10^{-6}$	$1.360\cdot 10^{-6}$
dpgmEDPG23	$1.359\cdot 10^{-3}$	$1.359\cdot 10^{-3}$	$1.359\cdot 10^{-3}$	$1.359\cdot 10^{-3}$
dpgmEPhosiDPG3	$6.090\cdot 10^{-8}$	$6.090\cdot 10^{-8}$	$6.090\cdot 10^{-8}$	$6.090\cdot 10^{-8}$
dpgmEPhosi	$7.310\cdot 10^{-4}$	$7.310\cdot 10^{-4}$	$7.310\cdot 10^{-4}$	$7.310\cdot 10^{-4}$
PG2	$1.400\cdot 10^{-2}$	$1.400\cdot 10^{-2}$	$1.400\cdot 10^{-2}$	$1.400\cdot 10^{-2}$
PEP	$1.700\cdot 10^{-2}$	$1.700\cdot 10^{-2}$	$1.700\cdot 10^{-2}$	$1.700\cdot 10^{-2}$
PYR	$7.700\cdot 10^{-2}$	$7.700\cdot 10^{-2}$	$7.700\cdot 10^{-2}$	$7.700\cdot 10^{-2}$
LAC	1.100	1.100	1.100	1.100
ADO	$1.200\cdot 10^{-3}$	$1.200\cdot 10^{-3}$	$1.200\cdot 10^{-3}$	$1.200\cdot 10^{-3}$
INO	$1.000\cdot 10^{-3}$	$1.000\cdot 10^{-3}$	$1.000\cdot 10^{-3}$	$1.000\cdot 10^{-3}$
akE	$1.810\cdot 10^{-4}$	$1.810\cdot 10^{-4}$	$1.810\cdot 10^{-4}$	$1.810\cdot 10^{-4}$
akEADO	$2.050\cdot 10^{-5}$	$2.050\cdot 10^{-5}$	$2.050\cdot 10^{-5}$	$2.050\cdot 10^{-5}$
akEADOMgATP	$1.000\cdot 10^{-7}$	$1.000\cdot 10^{-7}$	$1.000\cdot 10^{-7}$	$1.000\cdot 10^{-7}$
akEAMPMgADP	$4.880\cdot 10^{-5}$	$4.880\cdot 10^{-5}$	$4.880\cdot 10^{-5}$	$4.880\cdot 10^{-5}$
akEAMP	$3.000\cdot 10^{-7}$	$3.000\cdot 10^{-7}$	$3.000\cdot 10^{-7}$	$3.000\cdot 10^{-7}$
akEAMPADO	$1.000\cdot 10^{-6}$	$1.000\cdot 10^{-6}$	$1.000\cdot 10^{-6}$	$1.000\cdot 10^{-6}$
akET	$4.530\cdot 10^{-5}$	$4.530\cdot 10^{-5}$	$4.530\cdot 10^{-5}$	$4.530\cdot 10^{-5}$
akETADO	$1.000\cdot 10^{-8}$	$1.000\cdot 10^{-8}$	$1.000\cdot 10^{-8}$	$1.000\cdot 10^{-8}$
IMP	$1.000\cdot 10^{-2}$	$1.000\cdot 10^{-2}$	$1.000\cdot 10^{-2}$	$1.000\cdot 10^{-2}$
PRPP	$5.000\cdot 10^{-3}$	$5.000\cdot 10^{-3}$	$5.000\cdot 10^{-3}$	$5.000\cdot 10^{-3}$
ADE	$1.035\cdot 10^{-3}$	$1.035\cdot 10^{-3}$	$1.035\cdot 10^{-3}$	$1.035\cdot 10^{-3}$
NADP	$2.000\cdot 10^{-4}$	$2.000\cdot 10^{-4}$	$2.000\cdot 10^{-4}$	$2.000\cdot 10^{-4}$
g6pdER	$5.920\cdot 10^{-6}$	$5.920\cdot 10^{-6}$	$5.920\cdot 10^{-6}$	$5.920\cdot 10^{-6}$
g6pdERNADP	$8.270\cdot 10^{-8}$	$8.270\cdot 10^{-8}$	$8.270\cdot 10^{-8}$	$8.270\cdot 10^{-8}$
g6pdERNADPG6P	$7.780\cdot 10^{-8}$	$7.780\cdot 10^{-8}$	$7.780\cdot 10^{-8}$	$7.780\cdot 10^{-8}$
g6pdERNADPHGL6P	$5.300\cdot 10^{-14}$	$5.300\cdot 10^{-14}$	$5.300\cdot 10^{-14}$	$5.300\cdot 10^{-14}$
GL6P	$1.754\cdot 10^{-3}$	$1.754\cdot 10^{-3}$	$1.754\cdot 10^{-3}$	$1.754\cdot 10^{-3}$

g6pdERNADPH	$1.600 \cdot 10^{-13}$	$1.600 \cdot 10^{-13}$	$1.600 \cdot 10^{-13}$	$1.600 \cdot 10^{-13}$
NADPH	$6.580 \cdot 10^{-2}$	$6.580 \cdot 10^{-2}$	$6.580 \cdot 10^{-2}$	$6.580 \cdot 10^{-2}$
GO6P	$3.748 \cdot 10^{-2}$	$3.748 \cdot 10^{-2}$	$3.748 \cdot 10^{-2}$	$3.748 \cdot 10^{-2}$
RU5P	$4.937 \cdot 10^{-3}$	$4.937 \cdot 10^{-3}$	$4.937 \cdot 10^{-3}$	$4.937 \cdot 10^{-3}$
R5P	$1.267 \cdot 10^{-2}$	$1.267 \cdot 10^{-2}$	$1.267 \cdot 10^{-2}$	$1.267 \cdot 10^{-2}$
X5P	$1.478 \cdot 10^{-2}$	$1.478 \cdot 10^{-2}$	$1.478 \cdot 10^{-2}$	$1.478 \cdot 10^{-2}$
S7P	$2.399 \cdot 10^{-2}$	$2.399 \cdot 10^{-2}$	$2.399 \cdot 10^{-2}$	$2.399 \cdot 10^{-2}$
E4P	$5.075 \cdot 10^{-3}$	$5.075 \cdot 10^{-3}$	$5.075 \cdot 10^{-3}$	$5.075 \cdot 10^{-3}$
HX	$2.000 \cdot 10^{-3}$	$2.000 \cdot 10^{-3}$	$2.000 \cdot 10^{-3}$	$2.000 \cdot 10^{-3}$
R1P	$6.000 \cdot 10^{-2}$	$6.000 \cdot 10^{-2}$	$6.000 \cdot 10^{-2}$	$6.000 \cdot 10^{-2}$
MgAMP	$4.509 \cdot 10^{-1}$	$4.509 \cdot 10^{-1}$	$4.509 \cdot 10^{-1}$	$4.509 \cdot 10^{-1}$
MgDPG23	1.908	1.908	1.908	1.908

A.4 The Prokaryotic auto-regulatory Gene Network

The Prokaryotes Gene expression Network (PGN) [460] is a simple example of gene regulation mechanism, whereby a gene (DNA) coding for a protein (P) is inhibited by binding with a dimer of the protein itself (DNA:P2). The reaction-based model of the PGN consists of 8 reactions among 5 chemical species. Gene expression represents a good example of the stochastic phenomenon in biological systems, since the transcriptional regulators appear in few copies, making possible to describe the binding and release of the regulators in probabilistic terms. Table A.7 describes the reactions composing this simple model. The initial condition considered in this work corresponds to 500 molecules of the DNA species, while the amount of all other species is initially equal to zero.

Table A.7 List of the reactions of the PGN model.

No.	**Reagents**	**Products**	**Constant**
R_1	DNA + P_2	DNA:P_2	0.1
R_2	DNA:PP_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

A.5 The Ras/cAMP/PKA pathway

The Ras/cAMP/PKA signaling pathway in yeast *Saccharomyces cerevisiae* is fundamental in regulating cell growth and proliferation in response to nutritional sensing and stress conditions [383, 486]. In yeast, normal activity of the protein kinase A (PKA) directly affects cellular growth and cell cycle progression. Indeed, a reduction of PKA activity, related to a decrease in intracellular cyclic adenine mono-phosphate (cAMP), leads to growth and cell cycle arrest. cAMP is synthesized by the adenylate cyclase Cyr1 and induces the activation of the cAMP-dependent PKA protein. The adenylate cyclase activity is controlled by Ras and Gpa2 proteins. Ras proteins cycle between an inactive state and an active state, positively regulated by Cdc25—a Guanine Nucleotide Exchange Factor (GEF)—and negatively regulated by Ira1 and Ira2. Finally, two phosphodiesterase (Pde1 and Pde2) constitute the major feedback mechanisms in the pathway and are able to degrade cAMP.

The list of the reactions that describe the interactions among the species composing the Ras/cAMP/PKA pathway and the values of the associated constants are depicted in Table A.8. More details can be found in [44, 75, 339]. In Table A.9 are shown the initial molecular amounts whose value is different to 0.

Table A.8 List of the reactions included in the RBM of the Ras/cAMP/PKA pathway.

No.	Reagents	Products	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	0.01
R_{10}	Ras2-GTP-Ira2	Ras2-GDP + Ira2	0.25
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	0.1

R_{19}	(3cAMP)-PKA	cAMP + (2cAMP)-PKA	0.1
R_{20}	(2cAMP)-PKA	cAMP + cAMP-PKA	0.1
R_{21}	cAMP-PKA	cAMP + PKA	0.1
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	0.75
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	0.1
R_{28}	cAMP-Pde1p	cAMP + Pde1p	0.1
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	0.01
R_{36}	Ira2 + C	Ira2p + C	$1.0 \cdot 10^{-3}$
R_{37}	Ras2-GTP + Ira2p	Ras2-GTP-Ira2p	1.25
R_{38}	Ras2-GTP-Ira2p	Ras2-GDP + Ira2p	2.5
R_{39}	Ira2p	Ira2	10.0

Table A.9 Initial molecular amounts of the RBM of Ras/cAMP/PKA pathway.

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	$2.4 \cdot 10^7$

Notice that the amounts of the species GDP, GTP and ATP are kept constant during the simulation. According to the data presented in [151] and described in [44, 75], the values of the molecular amounts of the species are expressed as number of molecules per cell.

A.6 The Schlögl model

The Schlögl model [451, 465] [342, 354] represents one of the simplest prototypes describing biochemical systems characterized by bistability. This phenomenon consists in having two different stable steady states that can be reached in response to some chemical signaling (for further details, see [94, 345, 464]). As shown in Table A.10, this model is composed of 3 reactions that involves 4 molecular species. The initial molecular concentrations of the 4 species are listed in Table A.11.

Table A.10 List of the reactions of the Schlögl model.

No.	**Reagents**	**Products**	**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_2	X	B	3.5

Table A.11 Initial molecular amounts of the Schlögl model.

Species	**Initial amount**
A	$1 \cdot 10^5$
B	$2 \cdot 10^5$
X	250

Notice that the amounts of the species A and B are kept constant during the simulation. All the values of the molecular amounts of the species are expressed as number of molecules per cell.