Doctoral Thesis

Optimization in computational systems biology via high performance computing techniques

David Rodríguez Penas

2017

UNIVERSIDADE DA CORUÑA

Optimization in computational systems biology via high performance computing techniques

David Rodríguez Penas

Doctoral Thesis

March 2017

PhD Advisors:

Julio Rodríguez Banga

Patricia González Gómez

Ramón Doallo Biempica

PhD Program in Information Technology Research

UNIVERSIDADE DA CORUÑA

Dr. Julio Rodríguez Banga
Profesor de investigación
Instituto de Investigacións Mariñas
Consejo Superior de Investigaciones Científicas (CSIC).

Dra. Patricia González Gómez
Profesora Titular de Universidade
Dpto. Enxeñaría de Computadores
Universidade de A Coruña

Dr. Ramón Doallo Biempica
Catedrático de Universidade
Dpto. de Enxeñaría de Computadores
Universidade de A Coruña

CERTIFICAN

Que a presente memoria titulada "*Optimization in computational systems biology via high performance computing techniques*" foi realizada por D. David Rodríguez Penas baixo nosa dirección no Departamento de Enxeñaría de Computadores da Universidade da Coruña, e conclúe así a Tese Doutoral que presenta para optar ao grado de Doutor en Enxeñaría Informática coa Mención de Doutor Internacional.

En A Coruña, a 30 de Marzo do 2017

Fdo.: Julio Rodríguez Banga
Director da Tese Doutoral

Fdo.: Patricia González Gómez
Directora da Tese Doutoral

Fdo.: Ramón Doallo Biempica
Director da Tese Doutoral

Fdo.: David Rodríguez Penas
Autor da Tese Doutoral

A Tamara e a eses sobriños
que están por vir: Sergio e Marinha.

Agradecementos

En primeiro lugar, gustaríame especialmente agradecer ós meus directores Julio, Patricia e Ramón, tanto a oportunidade que me deron para desenvolver a presente tese, como a dedicación, axuda e apoio que me ofreceron durante todos estes anos. Foi un honor traballar ó voso lado, e o máis prezado que me levo desta tese é tódalas cousas que aprendín convosco.

Por outro lado, tamén quero recoñecer a tódolos meus compañeiros do Grupo de Enxeñaría de Procesos do Instituto de Investigacións Mariñas, toda a axuda e apoio que me ofreceron durante todo este tempo. Quedo moi agradecido a todos vós pola convivencia do día a día, o compañeirismo, e os bos momentos vividos.

Tamén quero facer referencia a toda a axuda recibida por: David Henriques, compañeiro do grupo co que máis traballei e que me proporcionou os problemas de bioloxía de sistemas a optimizar nesta tese; ó profesor Jose Egea, tanto polas súas contribucións durante a creación dos métodos propostos no Capítulo 3, como polo seu asesoramento en toda a estadística utilizada na presente tese; e tamén ó profesor Julio Sáez, tanto pola colaboración durante a elaboración do Capítulo 4, como a estupenda acollida e trato recibido por parte del e dos membros do seu grupo durante a miña estancia en Aquisgrán.

Quero engadir os meus agradecementos ó Grupo de Arquitectura de Computadores da Universidade da Coruña, e ó Departamento de Informática do Instituto de Investigacións Mariñas pertencente ao CSIC en Vigo, polo uso da súa infraestrutura e todo o soporte recibido.

E sobre todo, máis alá do terreo académico, teño que dar as grazas a miña familia por todo ese ánimo e cariño que me deron desinteresadamente durante todo este tempo. Primeiro a miña Tamara, máximo apoio dende o inicio e até o final desta aventura; ti fuches e serás a miña máxima motivación. Tamén ós meus pais, por padecer as miñas frustracións e aledarse dos meus logros; sempre serán os meus mellores mestres e co seu apoio incondicional todo é moito máis fácil. E por último, ós meus irmáns, os cales durante toda a miña vida son e serán un exemplo a seguir.

Por último, agradezo ós seguintes organismos o financiamento e apoio a este traballo: *Ministerio de Economía e Competitividade*, a través dos proxectos DPI2011-28112-C04, DPI2014-55276-C5, TIN2013-42148-P e TIN2016-75845-P (todos cofinanciados con fondos FEDER), e do programa de axudas para a formación de persoal de investigación FPI (Ref: BES-2012-059201, vinculada ó proxecto DPI2011-28112-C04); *Xunta de Galicia*, polas axudas para a consolidación e estruturación de unidades de investigación competitivas (Redes ref. R2014/041 e ref. R2016/045, e os Grupos de Referencia Competitiva GRC2013/055); *Centro de Supercomputación de Galicia (CESGA)* e *European Bioinformatics Institute (EBI)*, polo acceso ós seus recursos de computación; e *Microsoft Research*, polo acceso á plataforma Microsoft Azure a través dunha conta patrocinada.

Resumo

O obxectivo da bioloxía de sistemas computacional é xerar coñecemento sobre complexos sistemas biolóxicos a través da combinación de datos experimentais con modelos matemáticos e técnicas avanzadas de computación. O desenvolvemento de modelos dinámicos (cinéticos), tamén coñecidos como enxeñería inversa, é un dos temas chave nesta área. Nos últimos anos, moitas investigacións centráronse no escalado destes modelos, facendo da estimación dos parámetros destes modelos, tamén coñecida como calibración de modelos, unha tarefa complexa. Esa complexidade require o uso de ferramentas e métodos eficientes para acadar bos resultados nun tempo cálculo razoable. En xeral, para resolver este tipo de problemas úsanse métodos de optimización global, e en particular as metaheurísticas xurdiron como métodos eficientes para resolver os problemas máis custosos. Con todo, para a maioría das aplicacións reais, as metaheurísticas aínda requiren moito tempo de cálculo para obter resultados aceptábeis.

Nesta tese preséntase o deseño, implementación e avaliación de novas metaheurísticas paralelas, especializadas en resolver problemas de estimación de parámetros dentro do contexto da bioloxía de sistema. En concreto, propóñense novas metaheurísticas baseadas nos algoritmos de avaliación diferencial e de procura dispersa. As novas propostas teñen como obxectivo acadar un equilibrio entre as capacidades de exploración e explotación dos algoritmos. Ademais, demostran como a cooperación entre procuras concorrentes mellora o comportamento dos algoritmos, mellorando a calidade das solucións e diminuíndo o tempo de execución. Tamén estudáronse estratexias adaptativas para aumentar a robustez das propostas. Na avaliación usáronse tanto arquitecturas HPC tradicionais como novas infraestruturas na nube. Obtivéronse moi bos resultados con problemas de optimización de grande dimensión e complexidade.

Resumen

El objetivo de la biología de sistemas computacional es generar conocimiento sobre complejos sistemas biológicos mediante la combinación de datos experimentales con modelos matemáticos y técnicas avanzadas de computación. El desarrollo de modelos dinámicos (cinéticos), también conocido como ingeniería inversa, es uno de los temas clave en este campo. En los últimos años, ha surgido un gran interés en el escalado de estos modelos cinéticos, haciendo de la estimación de parámetros, también conocida como calibración de modelos, una tarea con una gran dificultad, que requiere el uso de herramientas y métodos eficientes para alcanzar buenos resultados en un tiempo razonable. En general, para resolver estos problemas se usan métodos de optimización global. En concreto, las metaheurísticas surgen como algoritmos eficientes a ser utilizado en los problemas más complejos. Sin embargo, en la mayoría de las aplicaciones reales, las metaheurísticas todavía requieren mucho tiempo de cálculo para obtener resultados aceptables.

Esta tesis presenta el diseño, implementación y evaluación de nuevas metaheurísticas paralelas, especializadas sobretodo en resolver problemas de estimación de parámetros en biología de sistemas. En concreto, se proponen nuevas metaheurísticas basadas en los algoritmos de evolución diferencial y de búsqueda dispersa. Las nuevas propuestas tienen como objetivo lograr un equilibrio entre las capacidades de exploración y explotación de los algoritmos. Además, demuestran como la cooperación entre búsquedas concurrentes mejora el comportamiento del algoritmo, mejorando la calidad de las soluciones y disminuyendo el tiempo de ejecución. También se han estudiado estrategias adaptativas para aumentar la robustez de las propuestas. Para la evaluación se han usado tanto arquitecturas HPC tradicionales como nuevas infraestructuras en la nube. Se han obtenido muy buenos resultados en problemas de gran dimensión y complejidad.

Abstract

The aim of computational systems biology is to generate new knowledge and understanding about complex biological systems by combining experimental data with mathematical modeling and advanced computational techniques. The development of dynamic models (also known as reverse engineering) is one of the current key issues in this area. In recent years, research has been focused on scaling-up these kinetic models. In this context, the problem of parameter estimation (model calibration) remains a very challenging task. The complexity of the underlying models requires the use of efficient solvers to achieve adequate results in reasonable computation times. Global optimization methods are used to solve these types of problems. In particular, metaheuristics have emerged as an efficient way of solving these hard global optimization problems. However, in most realistic applications, metaheuristics still require a large computation time to obtain acceptable results.

This Thesis presents the design, implementation and evaluation of novel parallel metaheuristics with the focus on parameter estimation problems in computational systems biology. In particular, we propose new cooperative metaheuristics based on the well known Differential Evolution and Scatter Search algorithms. The design of the novel approaches aim to achieve a proper balance between exploration (global search) and exploitation (local search) abilities. We show how the cooperation between parallel searches improves the behavior of the individual optimizers, improving the quality of the obtained solutions while decreasing the time-to-solution. We also explore adaptive strategies in order to increase the robustness of the algorithms. We present encouraging results for the proposed metaheuristics considering very challenging large-scale benchmark problems. Both traditional high performance computing (HPC) parallel and distributed architectures and new cloud infrastructures have been used to evaluate the proposals.

Preface

Many key problems in computational systems biology and bioinformatics can be formulated and solved using a global optimization framework. The complexity of the underlying mathematical models requires the use of efficient solvers in order to obtain satisfactory results in reasonable computation times. Metaheuristics are popular stochastic methods which are able to locate the vicinity of the global solution without having to explore all the search space, reducing the number of evaluations and, thus, the computational time. However, these stochastic algorithms still require excessive computational effort in many realistic applications, such as those considered in this Thesis, where we try to solve non-linear programming problems (NLP) and mixed-integer non linear problems (MINLP) subject to nonlinear dynamic equality and inequality constraints, a very complex task due to the multi-modal and non-convex nature of these optimization processes.

Current multiprocessor infrastructures such as computational clusters, supercomputers, clouds facilities or GPUs, and, furthermore, classical parallelization strategies such as MPI or openMP, offer great opportunities to improve the performance of classical metaheuristics. The parallelization of metaheuristics pursues one or more of the following goals: increase the size of the problems that can be solved, speed-up the computations, and/or attempt a more thorough exploration of the solution space. However, achieving an efficient parallelization of metaheuristics is usually a complex task, since the search for new solutions depends on previous iterations of the algorithm, which not only complicates the parallelization itself but also limits its scalability.

This Thesis proposes and evaluates different distributed metaheuristics, modified through high performance computing (HPC) techniques, and applied to address NLP or MINLP problems in current challenging optimization processes within the field of computational systems biology.

Work methodology

This Thesis follows a classical approach in scientific and technological research: analysis, design, implementation and evaluation. Thus, the Thesis starts with the analysis of the importance of global optimization processes in computational systems biology, in general, and parameter estimation problems, in particular; the state-of-the-art of metaheuristics applied to these kinds of problems; and the feasibility and impact analysis of using HPC solutions.

The first proposed parallel metaheuristic is based on an evolutionary method called Differential Evolution (DE), which has received a lot of attention during the last decade. In this Thesis, we present several enhancements to DE based on the introduction of additional algorithmic steps and the exploitation of parallelism. In particular, we propose an asynchronous parallel implementation of DE which has been extended with improved heuristics to exploit the specific structure of parameter estimation problems in computational systems biology. The proposed method is evaluated with different types of benchmarks problems, obtaining excellent results both in terms of quality of the solution and regarding speedup and scalability.

Then, a novel distributed metaheuristic is proposed, extending in several ways another popular algorithm, enhanced Scatter Search (eSS). We propose a self-adaptive asynchronous Cooperative enhanced Scatter Search (saCeSS) based on the parallel execution of different eSS threads and the asynchronous cooperation between them, the exchange of information being driven by quality of the solution obtained in each process, rather than by an elapsed time. This method incorporates several new key mechanisms: asynchronous cooperation between parallel processes; coarse and fine-grained parallelism; and self-tuning strategies, where the different settings of the metaheuristic change in execution time depending of the successes and failures of each distributed process. Several challenging parameter estimation problems from the domain of computational systems biology are used to assess the efficiency of the proposal, obtaining encouraging results in the scalability, robustness and performance of the method.

Furthermore, a set of modifications are applied to the proposed saCeSS algorithm to handle MINLP and mixed-integer dynamic optimization (MIDO) problems, two extremely challenging classes of problems. The new proposal obtains a good scalability and an important reduction in the distribution dispersion of the achieved results for these problems.

Finally, besides the evaluation in local clusters, the proposed techniques have also been assessed in a cloud infrastructure, the Azure Microsoft public cloud. Thus, the results obtained can be particularly useful, not only for the computational systems biology community, but also for those interested in the potential of cloud frameworks and platforms for developing metaheuristic methods in global optimization problems in general.

Structure of the Thesis

The Thesis is organized into five chapters:

- Chapter 1 summarizes background information pertinent to the research discussed in the remainder of this Thesis. This chapter describes basic concepts about global optimization methods in general, parameter estimation problems in particular, metaheuristics, and HPC architectures and programming models.
- Chapter 2 presents an improved DE algorithm designed to solve complex optimization problems within the field of parameter estimation problems in computational systems biology, since the improved local search is implemented by means of several heuristics which exploit the structure of these kinds of problems. The chapter describes the proposed method that also improves the global search through an asynchronous solution based on a cooperative island-model.
- Chapter 3 describes the novel parallel metaheuristic named saCeSS, based on an enhanced Scatter Search (eSS) method. Besides a coarse and fine-grained parallelization, this new method incorporates self-tuning strategies

during execution time, which results in a key mechanism to achieve a good scalability in very difficult problems. The chapter also provides an exhaustive evaluation of the performance obtained with the proposed saCeSS method, and a comparison with other parallel implementations.

- Chapter 4 focuses on applying saCeSS method to MINLP problems. The chapter describes the required modifications to be performed in the method with the aim to handle these specific kinds of optimization problems. It also provides an evaluation of the scalability and the robustness of the algorithm in very challenging case studies.

- Chapter 5 explores and presents the performance evaluation of the previous proposed methods in a cloud infrastructure, comparing it with the results obtained in local clusters. Additionally, a preliminary comparison of the MPI solutions proposed, that are HPC oriented, with other implementations using throughput oriented computing models, like Spark, is also shown in this chapter.

Finally, the work is concluded by summarizing the main contributions of this Thesis and the future research lines that can be derived from it.

Funding and Technical Means

The necessary means to carry out this Thesis have been the following:

- Working material, human and financial support primarily by the (Bio)Process Engineering Group at IIM-CSIC and by the Computer Architecture Group of the University of A Coruña, along with a Fellowship funded by the Spanish Ministerio de Economía y Competitividad under the FPI programme.

- Access to bibliographical material through the libraries of the University of A Coruña and IIM-CSIC library and archives network.

- Additional funding through the following research projects:

- The Spanish Ministerio de Economía y Competitividad and the FEDER, under Projects DPI2011-29112-C04 and DPI2014-55276-C5 (Bioprocess Engineering Group) and Projects TIN2013-42148-P and TIN2016-75845-P (Computer Architecture Group).
- Regional funding by the Galician Government (Xunta de Galicia) under the Consolidation Program of Competitive Research Groups: Computer Architecture Group, ref. GRC2013/055, and Network of Cloud and Big Data Technologies for HPC, Ref. R2014/041, and Network of Cloud and Big Data Technologies for HPC (II), ref. R2016/045.
- Microsoft Research through a sponsored Azure account.

▪ Access to clusters, supercomputers and cloud computing platforms:

- Pluton cluster (Computer Architecture Group, University of A Coruña, Spain): 16 nodes powered by two octa-core Intel Xeon E5-2660 CPUs with 64 GB of RAM, connected through an InfiniBand FDR network.
- NEMO cluster (Bioprocess Engineering Group, The Spanish National Research Council, Spain): 4 nodes powered by two quadcore Intel Xeon E5420 CPUs with 16 GB of RAM, 8 nodes powered by two quadcore Intel Xeon E5520 CPUs with 24 GB of RAM, and three nodes powered with two deca-core Intel Xeon E5-2650 CPUs with 30GB of RAM, connected through a Gigabit Ethernet network.
- SVG Linux cluster (Galicia Supercomputing Center, CESGA, Spain): 18 nodes powered by two dodeca-core Intel Haswell E5-2680 CPUs with 64 GB of RAM, 10 nodes powered with one tetra-core Intel Haswell E5-1240 with 32 GB of RAM, and 8 nodes powered by two octa-core Intel Sandy Bridge E5-2670 with 64 GB of RAM, connected through an InfiniBand FDR network.
- EBI Cluster (European Bioinformatics Institute): 222 nodes powered by two octa-core Intel Xeon E5-2680 CPUs with 30GB of RAM, connected through a Gigabit Ethernet network.
- Microsoft Azure cloud platform. Several instance types have been used: (1) A-3 instances, with 4 cores with 7 GB of RAM per node; (2) intensive-compute A-11 instances, with 16 cores with 112 GB of RAM per node;

and (3) intensive-compute A-9 instances, also with 16 cores with 112 GB of RAM per node, that, in addition to the standard Azure network interface available to the other instances, features a second network interface for remote direct memory access (RDMA) connectivity.

- A three-month research visit to Joint Research Center for Computational Biomedicine (RWTH Aachen), Germany from 11/04/16 to 10/07/16. This research visit was funded by FPI visit program.

Contents

List of Tables

List of Figures

List of Algorithms

List of Abbreviations

aCeSS	asynchronous Cooperative enhanced Scatter Search
API	Application Programming Interface
asynPDE	asynchronous Parallel Differential Evolution
asynPDE_IH	asynchronous Parallel Differential Evolution with Improved Heuristics
CeSS	Cooperative enhanced Scatter Search
DE	Differential Evolution
DO	Dynamic Optimization
eSiPDE	enhanced Spark island-based Parallel Differential Evolution
eSS	enhanced Scatter Search
GO	Global Optimization
GPU	Graphic Processor Unit
HDFS	Hadoop Distributed File System
HPC	High Performance Computing
LP	Linear Programming
MIDO	Mixed-Integer Dynamic Optimization
MIMD	Multiple Instruction Multiple Data
MINLP	Mixed Integer Non-Linear Programming

MIOC	Mixed-Integer Optimal Control
MISQP	Mixed-Integer Sequential Quadratic Programming
MPI	Message-Passing Interface
MPP	Massively Parallel Processing
NL2SOL	Adaptive Nonlinear Least-Squares Algorithm
NLP	Non-Linear Programming
OC	Optimal Control
ODE	Ordinary Differential Equation
PGAS	Partitioned Global Address Space
RDD	Resilient Distributed Dataset
RDMA	Remote Direct Memory Access
saCeSS	self-adaptive Cooperative enhanced Scatter Search
saCeSS2	self-adaptive Cooperative enhanced Scatter Search version 2
seqDE	sequential classic version of Differential Evolution
seqDE_IH	sequential version of Differential Evolution with Improved Heuristics
SIMD	Simple Instruction Multiple Data
SiPDE	Spark island-based parallel Differential Evolution
SMP	Shared Memory Multiprocessors
SmsPDE	Spark-based master-slave Parallel Differential Evolution
SMT	Simultaneous Multi-Threading
SS	Scatter Search
SSP	Synthetic signaling pathway

synPDE	synchronous Parallel Differential Evolution
synPDE_IH	synchronous Parallel Differential Evolution with Improved Heuristics enabled
TS	Tabu Search
VLIW	Very Long Instruction Word
VTR	Value-To-Reach

Chapter 1

Background

This chapter presents some background related to the research presented in this Thesis. We start with a review of the state-of-the-art in high performance computing (HPC) architectures and programming models. Then, we present a brief review of optimization problems and methods in computational systems biology. Finally, a section on parallel metaheuristis aims to introduce the target of this Thesis, as well as the strategies followed in the design, implementation and evaluation of the proposals.

This chapter intends to provide a brief yet important context for the kind of problems that this Thesis aims to solve and the means to achieve this objective. Take note that the related work covering alternative solutions to the ones proposed in this Thesis is not included in this chapter, but rather presented in next chapters.

1.1. Introduction to high performance computing

Since the appearance of the first digital computers, computer engineers have targeted increasing the speed of computer operations as one of their primary objectives. Obtaining higher operating speeds may be achieved in different ways [93]: improving the technology used for the implementation of the computer components, modifying the logical design of the subsystems, or improving algorithms for problem solving. A different approach to attain this goal is *parallel processing* [119], whose main principle is to split the computational cost of a problem into a set of tasks that can be performed concurrently. High performance computing (HPC) can be described as the use of parallel processing for solving complex computational problems. The number of research fields demanding HPC solutions, such as climate science, high-energy physics, chemistry, bio-technology, etc., are continuously increasing [86].

Employing HPC as a research tool demands at least a basic understanding of the hardware concepts and software issues involved. The following subsections aim to give the reader an introduction to current HPC architectures and programming models. Also, the term cloud computing is introduced, focusing in the concept of *Infrastructure as a Service*, as a way of provisioning HPC resources.

1.1.1. Trends in HPC architectures

In early computer architectures, processor operation was very simple and strictly sequential. Soon, different approaches to parallel processing arose [93], by means of exploiting *Instruction-Level Parallelism* (ILP), *Data-Level Parallelism* (DLP), and *Thread-Level Parallelism* (TLP). In the ILP approach the various operations involved in executing a single instruction can be separated into different stages, overlapping the execution of instructions when they are independent of one another. Additionally, functional units can be also replicated, further enabling the execution of instructions in parallel, such as in the superscalar and VLIW (*Very Long Instruction Word*) architectures. However, the improvement in the performance with this kind of parallelism is limited. The DLP approach pursues parallelism by means of vector architectures, and more recently *graphic processor units* (GPUs), that apply a single instruction to a collection of data in parallel. The TLP approach, exploits thread concurrency either at a core level (such as simultaneous multithreading - *SMT*), or at a multicore level on a chip.

In the early XXI century, the trend to improve the performance of microprocessors focused in increasing the clock speed. At this time, microprocessors started to be limited by the *heat barrier*: switching and leakage power of several hundred-million transistor chips are so large that cooling becomes a primary effort. This power-performance issue was solved by means of *multicore* processors, i.e., systems

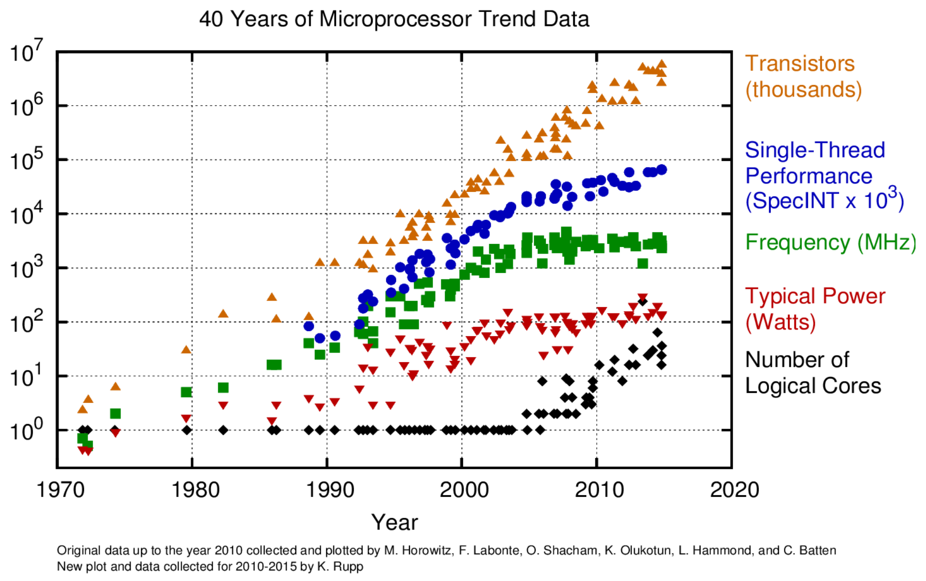
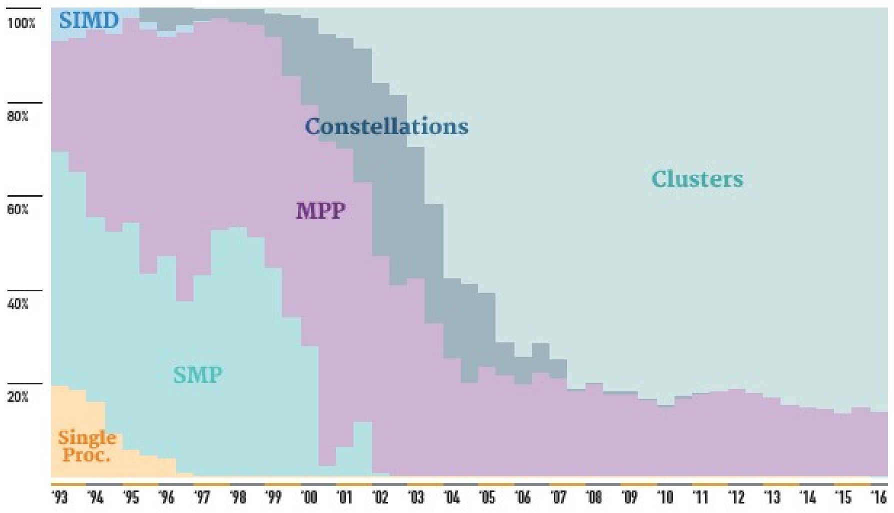


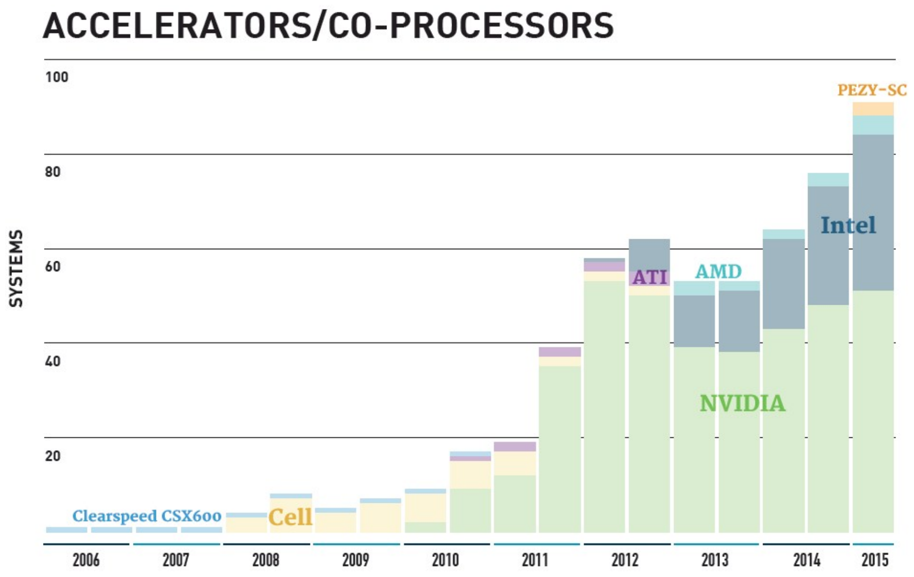
Figure 1.1: Microprocessors trend data over the past 40 years. Source: [110].

with several cores on a single socket. Figure 1.1 illustrates how gains in frequency and single-thread performance have stagnated in recent years. Thus, multicore processors have taken over, aiming to exploit the number of transistors in the chip, that still grows exponentially following Moore's law [142]. The multicore approach allows performance scaling without pursuing new clock frequency increments, which would exacerbate the heat barrier issue.

The bi-annual list of the world's fastest, most powerful supercomputers, the Top500 list [206], gives an interesting historical overview regarding HPC architectures and performance evolution. In the 1980s, vector supercomputing dominated HPC. The 1990s saw the rise of massively parallel processing (MPPs) and shared memory multiprocessors (SMPs). In turn, clusters of commodity and purpose-built processors dominated the previous decade. Today, these clusters are expanded with computational accelerators in the form of coprocessors, such as the Intel Xeon Phi or the GPUs. Figure 1.2 shows the evolution of the systems architecture in the Top500 list (Figure 1.2(a)), including a demonstration of the accelerator incursion (Figure 1.2(b)). Moreover, Figure 1.3 outlines the historical and projected performance development of the systems in the Top500 list. Just a few years ago, teraflops (10^{12} floating point operations/second) defined the state-of-the-art in ad-



(a) Architecture evolution.



(b) Accelerator incursion.

Figure 1.2: Development over time according to the Top500 list [206].

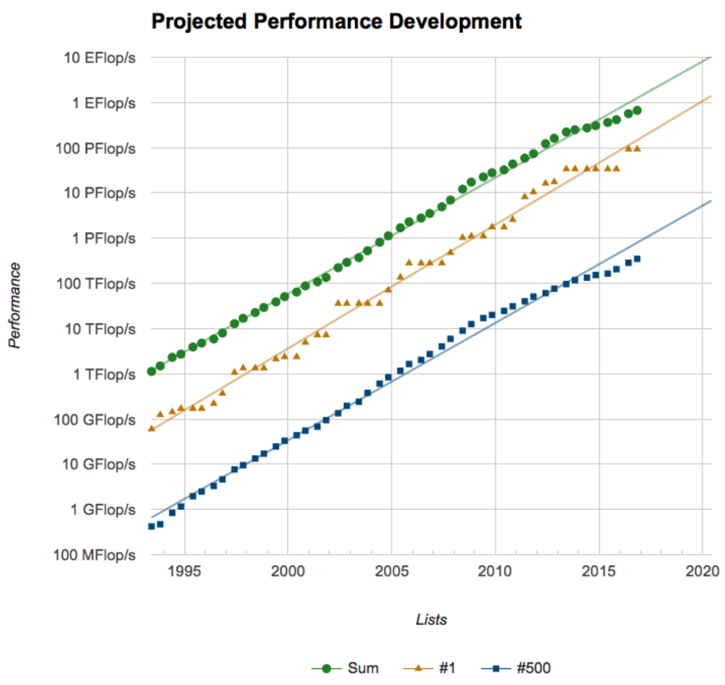


Figure 1.3: Supercomputing performance evolution according to the Top500 list [206].

vanced computing. Today, these values represent a desktop PC with a GPU or Xeon Phi accelerator. Supercomputing is now defined by multiple petaflops (10^{15} flops). If the projections hold we can expect an exaflop system at around 2020.

Most of the current parallel computational infrastructures are classified, following the Flynn taxonomy [73], as MIMD (*Multiple Instruction Multiple Data*) machines. There are a large variety of MIMD systems, that is to say, the way in which the processors communicate with the memory subsystem, that significantly determines the performance of a multiprocessor system, the usual way to classify multiprocessor systems [93]:

- *Shared-memory* computers. These are systems with multiple cores/processors, sharing among them the same physical address space. Figure 1.4 shows an

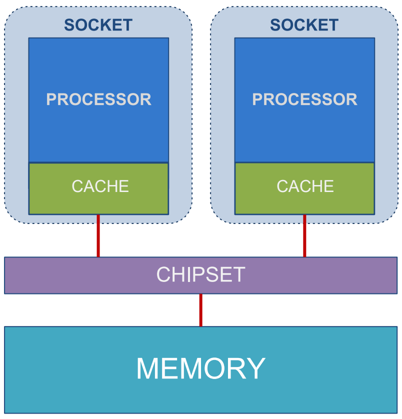


Figure 1.4: Example of shared-memory architecture. Adapted from [86].

example for this type of machine. The user is not concerned with where the data is stored, as there is only one memory accessed by all cores on an equal basis.

- *Distributed-memory* computers. This scheme has multiple cores, each of them with its own associated memory. Figure 1.5 shows a simplified block diagram of a distributed-memory parallel computer. The cores are connected by some network and may exchange data among their respective memories when required. In contrast to shared memory machines, the user must be aware of the location of the data and will have to move these data explicitly when needed.

- *Hierarchical* (hybrid) computers. Multiprocessors can also be designed neither of the shared-memory nor of the distributed-memory type but a mixture of both. That is, shared-memory building blocks are connected via a fast network (see Figure 1.6), configuring a global distributed-memory multiprocessor where each node has shared-memory properties. The concept is actually more generic and can also be used to describe any system with a mixture of different hardware layers. Examples are clusters built from nodes that contains, besides the multi-core processors, additional *accelerator hardware*, such as GPUs, FPGAs (*Field-programmable Gate Arrays*), or general computational accelerators.

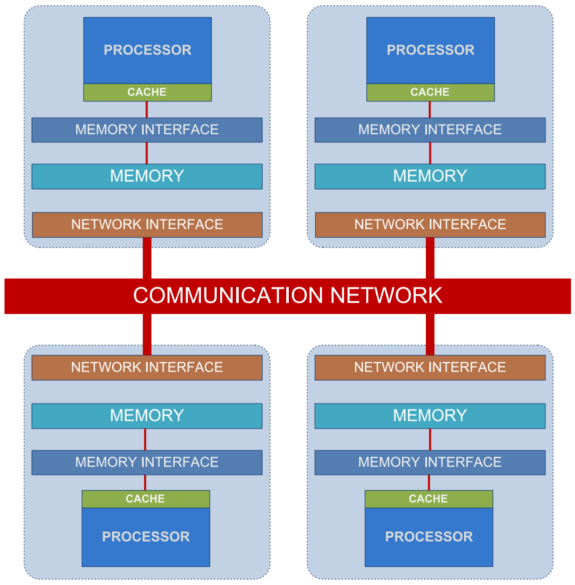


Figure 1.5: Example of distributed-memory architecture. Adapted from [86].

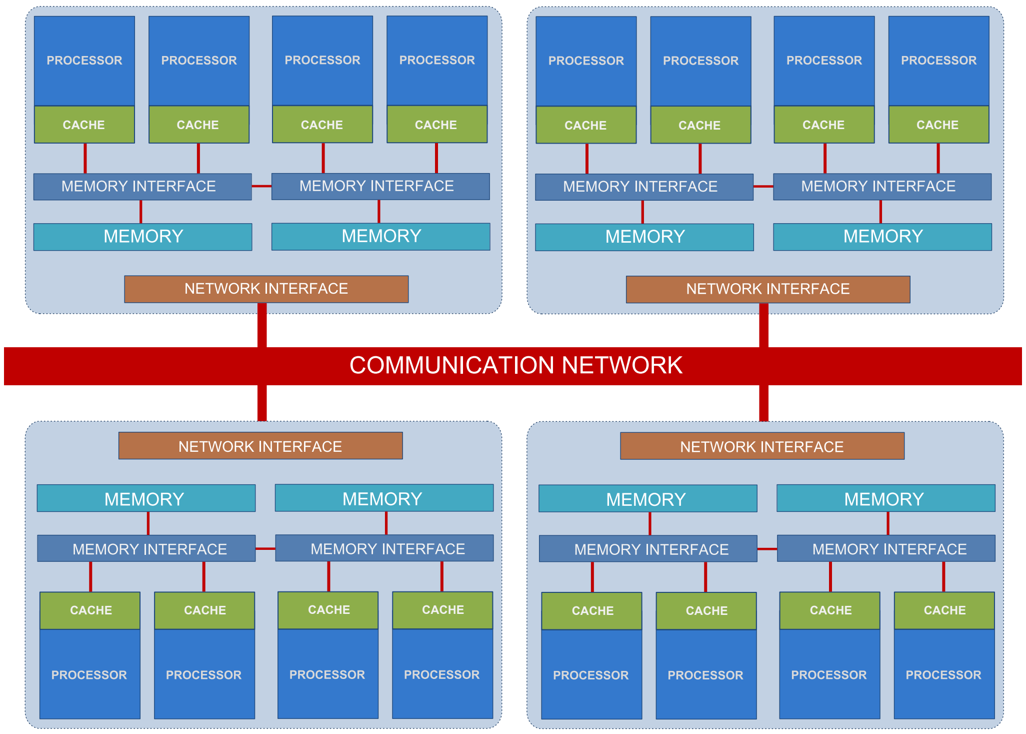


Figure 1.6: Example of hierarchical (hybrid) scheme. Adapted from [86].

1.1.2. Parallel programming models

Though the compiler and hardware work together to exploit instruction-level parallelism implicitly without the programmer's attention, the efficient programming of modern hierarchical computers requires the restructuring of the application so that it can exploit explicit parallelism at higher levels, such as threads, processes or programs. Sometimes this could be an easy task, however, in many instances, it requires a significant effort from programmers.

Parallel applications should be written following a programming model. The simplest case of parallel execution consists of the multiprogramming model, where several sequential programs are executed concurrently on different processors without any interaction among them. However, the most interesting case is that of the parallel programs, that consists of multiple tasks running on multiple processors requiring cooperation between them. Basically, the following are the most popular alternatives to develop a parallel program, nowadays:

- *Message-passing model:* The message-passing model consists of a set of processes that are able to communicate with each other by sending and receiving messages. In the message-passing model of parallel computation, the processes executing in parallel have separate address spaces. Communication occurs when a portion of one process's address space is copied, in a cooperative way, into another process's address space. This model provides the programmer with explicit control over the location of memory in a parallel program, specifically, the memory used by each process. This ability to manage memory location can allow the programmer to achieve high performance. However, the main drawback of message passing is that the programmer needs to pay attention to details such as data placement on memory and the ordering of communication. The Message-Passing Interface (MPI) [145, 161] is a library that allows developers to write robust and efficient parallel and distributed applications using the message-passing paradigm. MPI is, probably, the most widely used programming framework in the HPC community.
- *Shared-memory model:* Parallel programs running on shared memory systems are split into several processes, called *threads*, that share data related to a portion of their address space. The interaction between threads is performed

implicitly by reading and writing shared variables. Typically, each process can carry out the execution of a subset of iterations from a common loop, or, more generally, each process can get its tasks from a shared queue. Currently, openMP [203] is the de-facto standard for shared-memory parallel programming.

- *Data Parallel Programming Model:* This is a programming model inherited from SIMD (*Simple Instruction Multiple Data*, following the Flynn taxonomy [73]) machines. Currently it may also be referred to as Partitioned Global Address Space (PGAS) model. In this approach, address space is treated globally. Most of the parallel work focuses on performing operations on a data set. A set of tasks work collectively on the same data structure, however, each task works on a different partition of the same data structure. There are currently several relatively popular parallel programming implementations (at different levels of development) based on this model, such as Unified Parallel C (UPC) [210], Global Arrays [79], X10 [234], and Chapel [43].
- *Automatic Parallelization:* In this case, the compiler assumes all strategies and decisions, generating the parallel version of the original sequential code automatically. In general, the current automatic parallelisers provide good results when the codes to be parallelized are simple, with regular access patterns to data. However, the automatic parallelization of complex and/or irregular codes is an extremely hard task and there is not an efficient solution at the moment. Several research groups are working in this field, giving rise to interesting tools such as Paralax [211], Helix [39], KIR [12], TRACO [162], and Sambamba [195].

Besides the programming model, an important feature related to the design of parallel programs is the granularity. This is a way of measuring the degree of parallelism exploited by the system. This property represents the number of computations performed by processes without needing cooperation between them. Thus, in a *coarse* granularity parallelization, the computational load of the program is split into several tasks that usually requires a moderate number of communications between them. On the contrary, *fine* granularity parallelization is characterized by more intensive communication between processes, where usually relatively few instructions are performed without needed communications.

In this Thesis we have designed parallel algorithms to be executed in MIMD distributed-memory multiprocessors composed of shared-memory nodes. For code development C and Fortran languages have been used together with the MPI message passing routines and openMP directives, so that the codes are portable to most current machines and architectures. Depending on the algorithm at hand, either coarse grain parallel solutions (using the MPI library) or hybrids solutions where coarse grain strategies were combined with fine grain intensifications (using MPI+OpenMP) have been proposed.

1.1.3. Cloud Computing

Cloud computing [14,38] is the evolution of a collection of technologies that have been gathered together to redefine the approach for building an IT infrastructure. Nothing is essentially new in any of the technologies employed in cloud computing, since most of them have been already used. The cloud computing term describes a computing paradigm, where a large pool of systems are connected in private or public networks, to provide dynamically scalable infrastructure for application, data and file storage. To this end, clouds are built using virtualized infrastructure technology. Virtualization is the process of converting a physical IT resource into a virtual one. Thus, cloud computing follows a very fundamental principal of reusability of IT capabilities, relying on the sharing of various resources (e.g., networks, servers, storage, applications, and services). With the advent of this technology, the cost of computation, application hosting, content storage and delivery is reduced significantly.

A public cloud offers access to external users who are usually billed by consumption using the *pay-as-you-go* model. Cloud Providers offer services that can be grouped into three categories [100]: *Software as a Service* (SaaS), where a complete application is offered to the customer as a service on demand; *Platform as a Service* (PaaS), where a layer of software is encapsulated and offered as a service; and *Infrastructure as a Service* (IaaS), which provides basic storage and computing capabilities as standardized services over the network. Public IaaS cloud providers typically make huge investments in data centers and then hire it out, allowing consumers to avoid substantial capital investments and to obtain both cost-effective

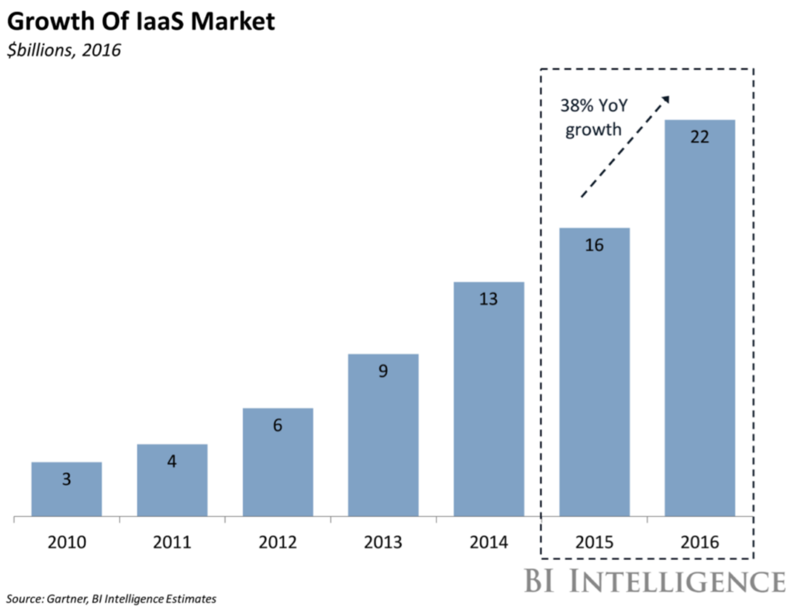


Figure 1.7: Growth of IaaS market. Source: [77]

and energy-efficient solutions [180]. IaaS accounts for less than ten percent of the cloud market in 2016 [77]. However, it was the fastest growing cloud-based service (see Figure 1.7), and it is expected to repeat the strong growth in 2017 as well.

Considering the demanding and dynamic nature of HPC applications, Cloud Computing technologies represent a powerful approach to managing technical computing resources [178]. The elastically scaling out to meet increased capacity demands is the obvious benefit of the cloud. Besides, other features make cloud computing an attractive option for meeting the needs of HPC applications. The cost savings in the cloud can be significant. The cloud supports rapid provisioning for particular workloads. The ability to rapidly provision new environments/clusters in minutes is key to the success and practicality of many HPC applications, compared to the time it can typically take to provision new hardware on-premise. Summarizing, combining scale and elasticity creates a capability for HPC cloud users that does not exist for centralized shared HPC resources. Each HPC user in the cloud can have access to their own set of HPC resources, such as compute, networking, and storage resources for their own specific applications with no need to share the resources with other users. They have zero queue time and can create systems architectures that their applications need.

In spite of the previous commented benefits, some challenges still remain for the adoption of cloud in HPC applications [178]. The most important are security and performance. Security remains a significant barrier to adoption, however the issue resides primarily in users' trust and perception rather than limitations in capability and architecture of various cloud platforms. Regarding performance, in the last decade, several researchers have studied the performance of HPC applications in cloud environments [63, 66, 70, 102, 147]. Most of these studies use classic MPI benchmarks to compare the performance of MPI on public cloud platforms. These works conclude that the lack of high-bandwidth, low-latency networks, as well as the virtualization overhead, has a large effect on the performance of HPC applications on the cloud. It is in response to these issues that some cloud providers, such as Amazon [11] or Microsoft Azure [139], have recently provided compute nodes which utilize hardware found in HPC clusters and that assert to be optimized for running HPC applications.

Programming frameworks in the cloud

New programming environments are being proposed to deal with large scale computations on the cloud. These new distributed frameworks provide high-level programming abstractions that simplify the development of distributed applications including implicit support for deployment, data distribution, parallel processing and run-time features like fault tolerance or load balancing.

From the new programming models that have been proposed to deal with large scale computations on cloud systems, MapReduce [53] is the one that has attracted more attention since its appearance in 2004. In short, MapReduce executes in parallel several instances of a pair of user-provided *map* and *reduce* functions over a distributed network of *worker* processes driven by a single *master*. Executions in MapReduce are made in batches, using a distributed filesystem (typically HDFS) to take the input and store the output. MapReduce has been applied to a wide range of applications, including distributed pattern-based searching, distributed sorting, graph processing, document clustering or statistical machine translation among others. However, when it comes to iterative algorithms MapReduce has shown serious performance bottlenecks [64] mainly because there is no way of efficiently reusing data or computation from previous iterations. New proposals, not based on MapReduce, like Spark [235] or Flink, which has its roots in Stratosphere [9], are designed from the very beginning to provide efficient support for iterative algorithms.

Spark provides a language-integrated programming interface to resilient distributed datasets (RDDs), a distributed memory abstraction for supporting fault-tolerant and efficient in-memory computations. According to [235] the performance of iterative algorithms can be improved by an order of magnitude when compared to MapReduce.

In Chapter 5 of this Thesis we explore the feasibility of deploying our experiments on clouds, specifically on the Microsoft Azure public cloud. A performance evaluation has been carried out, comparing the obtained results with those of the local clusters. Besides, a preliminary comparison of one of the metaheuristics proposed, that is HPC oriented, with other similar implementation using Spark, that is throughput oriented, is also performed.

1.2. Optimization in computational systems biology

Molecular biology has achieved great progress since the middle of the last century, produced by the advent of new technologies and techniques. These advances have allowed identifying and studying those components which are part of biological systems, such as genes, nucleic acids, proteins, etc. However, this knowledge is not enough to provide an understanding of the relationships between molecular components of the entire system, such as in the metabolic networks or cell signaling networks. Hence, *systems biology* arises to be in charge of accomplishing the study of biological systems as an entire system, studying those complex interplays among molecular components via the developing of mathematical models, which are analysed with the aim of obtaining biological predictions and new knowledge. The concept of systems biology has been widely used from year 2000 onwards [56, 114, 115, 233], being a very multidisciplinary field, where several techniques and methods from systems engineering, statistics, computational science, computational biology, and molecular biology, are combined to obtain results in areas such as genomic, bioinformatics or analysis of systems' dynamics.

The emergence of systems biology contributes to develop a system-level abstraction to understand biological systems, using new advances in software and computation which have enabled the creation and analysis of very useful biological models. In this Thesis, we focus on problems of *computational systems biology*, that is, a branch of systems biology that analyses biological data via computational techniques to obtain system-level approaches.

One of the main targets in systems biology is understanding the typical cell functions, such as division, differentiation, growth and apoptosis, which are temporal processes that can be handled as dynamic systems. Dynamical systems theory and control theory, commonly used in many branches of engineering and mathematics, can then be applied to these dynamical systems, so that they can be described by means of mathematical models. In particular, optimization plays a key role in many problems related to the modeling and design of biological systems [19]. Thus, we begin this section with an introduction to several basic concepts, related to optimization problems in computational systems biology applications, that can help readers unfamiliar with mathematical optimization. Then, we will describe the most popular global optimization methods to solve these optimization problems, focusing on metaheuristics.

1.2.1. Optimization problems

The optimization process consists in locating the best solution or *optimum* inside of a topology or *search space* described by one or more mathematical functions. Optimization problems [130, 196] can be defined through the couple (S, f), being S the set of all feasible solutions, and $f : S \to \mathbb{R}$ the *function cost* or *objective function*, obtaining a fitness value for each solution $s \in S$. The objective functions are subject to *constraints*, these being requirements that must be met, usually expressed as equalities and inequalities.

Each solution $s \in S$ contains a set of *decision variables*, which can be continuous, if they are represented by real numbers, or discrete, if they are represented by integer numbers. In many cases, there is a mix of continuous and integer decision variables. These variables can adopt different values within search space during the search of the optimum, determining the fitness value returned by objective function. The objective function is optimized (searching for a minimum or maximum) locating their global optimum: being $s^* \in S$, solution s^* is a global optimum if it has a better objective function than all the solutions of the search space, that is, $\forall s \in S, f(s^*) \leq f(s)$ (in minimization case).

Optimization problems can be categorized in different models, such as combinatorial optimization models, constraint satisfaction models, non-analytic models, or mathematical programming models. Those based on mathematical programming are the most popular ones, and can be further classified in linear and nonlinear models, depending if the objective function and the constraints were linear or not with respect to the decision variables.

In mathematics, when a function is lineal, the surface defined is a convex polyhedron, with a unique optimum solution (unimodality), i.e. with a unique maximum or minimum. However, when a function presents nonlinearities, that might imply nonconvexity, results in potential presence of multiple local solutions (multimodality) in the search space. Simple examples of unimodal and multimodal surfaces are shown in Figure 1.8.

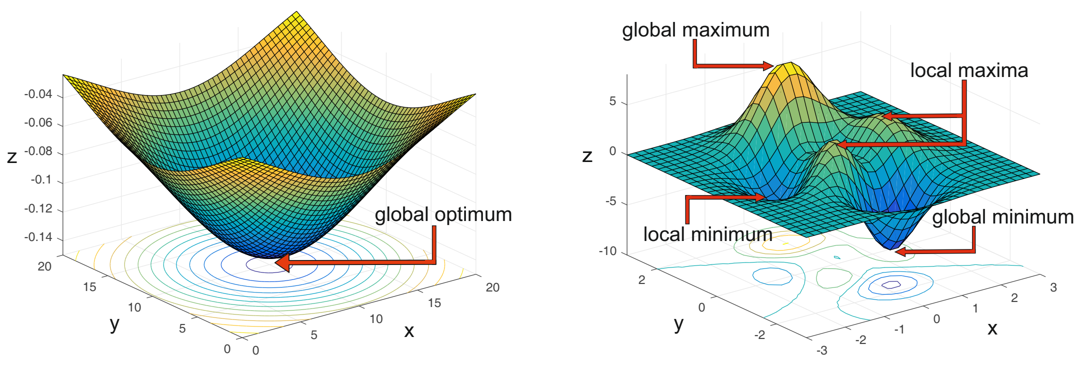
Mathematical programming problems can be classified (see Figure 1.9) according to the properties described by their functions, such as the existence of nonlinearities, the domain of the variables, or the presence of differential equations as constraints and time dependent decision variables.

Linear programming

Mathematical optimization problems based on *linear programming* (LP) can be stated as:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & A\,x = b \\ & x \geq 0 \end{array}$$

where x is a n-dimensional vector of decision variables, $c^T x$ is the objective function to minimize, c^T is the transpose of coefficient matrix of the cost function, A and b are, respectively, the coefficient matrix and vector of the constraints. Typically,



(a) Unimodal. (b) Multimodal.

Figure 1.8: Example of unimodal and multimodal surfaces. The z-coordinate of the surface represents the value of the objective function for each pair of decision.

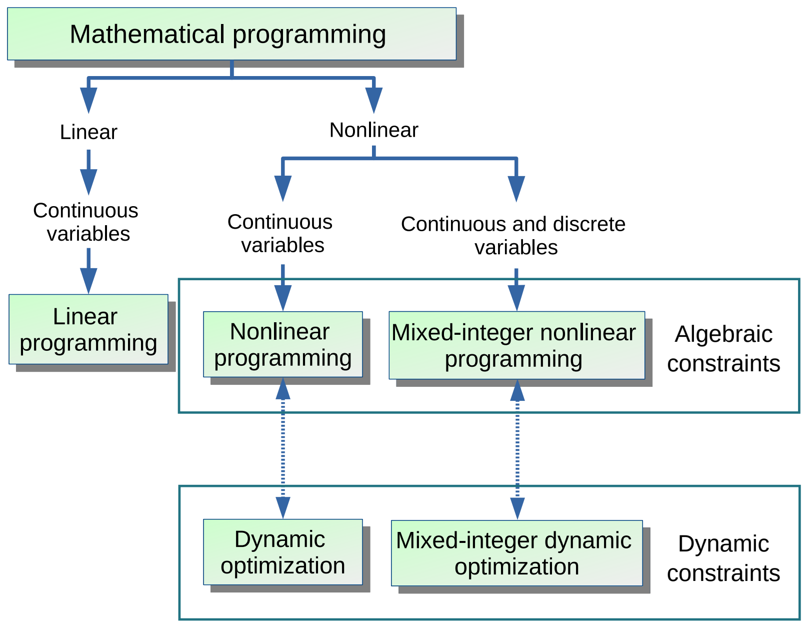


Figure 1.9: Classification of mathematical programming.

a LP problem presents a very simple scheme. The feasible region of the problem is convex, so that every local optimum represents a global optimum (unimodality), and the cost function is easy to solve.

Nonlinear programming

In models based on *nonlinear programming* (NLP), the complexity increases due to the nonlinearity of the objective function and constraints. A NLP problem consists in:

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & h_i(x) = 0,\ i\text{=1,2, ..., } m \\ & g_j(x) = 0,\ j\text{=1,2, ..., } p \\ & x \in S \end{array}$$

where x is a n-dimensional vector of continuous decision variables, containing values within the continuous subset S. Then, f is the cost function of the problem and the functions h_i and g_j are constraints. The presence of nonlinearities in the objective and constrains might imply nonconvexity, which can induce multimodality in the topology.

NLP problems are much more difficult to solve than LP problems, specially for large dimensions. The appropriate algorithms for solving LP problems, will obtain little success when applied to NLP problems, even for a medium or a small problem dimension. Due to non-linearity, non-convexity, multi-modality, and ill-conditioning in the topology described by the objective functions, many NLP problems are *NP-hard* (non-deterministic polynomial-time hard), and thus, very difficult to solve.

Mixed-integer nonlinear programming

When an optimization problem presents nonlinearity and the domain of the variables can be discrete or continuous, we talk about *mixed integer nonlinear programming* (MINLP) problems [35]. These problems can be described by:

$$\begin{aligned} \text{minimize} \quad & f_0(x,y) \\ \text{subject to} \quad & f_j(x,y) \leq 0,\ j = 1,2,...,p \\ & x \in \mathbb{Z}^n_+ \\ & y \in \mathbb{R}^m_+ \end{aligned}$$

where $f_0(x, y)$ is the objective function, f_j are a set of p constraints, x is a vector of decision discrete variables with dimension n, and y is a vector of decision continuous variables with dimension m. Besides, MINLP can be convex when all the functions f_j are convex, or non-convex otherwise. Both convex and non-convex MINLP problems are NP-hard. However, non-convex MINLPs are extremely hard to solve compared to NLP and convex MINLP problems.

Dynamic optimization problems

In addition to the problems described above, a new type of optimization is demanded to handle dynamic behaviors appearing in important phenomenas in many real-life applications. *Dynamic optimization* [21] problems (DO), also called optimal control (OC) problems, arises to obtain the optimal solution in systems whose dynamics are mapped by differential equations as constraints and time-dependent decision variables.

DO problems consider the computation of time-dependent conditions (controls, *stimuli*) and time-independent parameters so as to optimize (minimize or maximize) a performance index $J(\mathbf{x}, \mathbf{u})$ while satisfying a set of dynamic and algebraic constraints. Mathematically, it may formulated as follow:

$$J(\mathbf{x}, \mathbf{u}) = \Theta(\mathbf{x}(t_f)) + \int_{t_0}^{t_f} \Phi(\mathbf{x}(t), \mathbf{u}(t), t)dt \tag{1.1}$$

subject to:

$$d\mathbf{x}/d\mathbf{t} = \Psi(\mathbf{x}(t), \mathbf{u}(t), t) \tag{1.2}$$

$$\mathbf{h}(\mathbf{x}(t), \mathbf{u}(t)) = 0 \tag{1.3}$$

$$\mathbf{g}(\mathbf{x}(t), \mathbf{u}(t)) \leq 0 \tag{1.4}$$

$$\mathbf{x}_L \le \mathbf{x}(t) \le \mathbf{x}_U \tag{1.5}$$

$$\mathbf{u}_L \le \mathbf{u}(t) \le \mathbf{u}_U \tag{1.6}$$

where $J(\mathbf{x}, \mathbf{u})$ is the cost function, $d\mathbf{x}/d\mathbf{t}$ is a set of ordinary differential equality constraints, $\mathbf{x}$ is the vector of state variables with initial conditions $\mathbf{x}(t_0) = \mathbf{x}_0$, $\mathbf{u(t)}$ is the vector of real valued control variables, $\mathbf{h}$ and $\mathbf{g}$ a set of algebraic and inequality constraints, and $\mathbf{x}_L$, $\mathbf{x}_U$, $\mathbf{u}_L$, $\mathbf{u}_U$ correspond to the lower and upper bounds for the control variables and state parameters.

Mixed-integer dynamic optimization

When part of the decision variables of a DO problem includes discrete values, these formulations belong to the class of *mixed-integer dynamic optimization* (MIDO) problems, also called mixed-integer optimal control problem (MIOC). The general MIDO problem considers the computation of time dependent conditions (controls, *stimuli*), discrete decisions (binary or integer), and time-independent parameters, so as to minimize (or maximize) a performance index (cost functional) while satisfying a set of dynamic and algebraic constraints. In mathematical form, it is usually formulated as follows:

Find $\mathbf{u}(t)$, $\mathbf{i}(t)$, $\mathbf{p}$ and t_f so as to minimize (or maximize):

$$J = G_{t_f}(\mathbf{x}, \mathbf{u}, \mathbf{i}, \mathbf{p}, t_f) + \int_{t_0}^{t_f} F(\mathbf{x}(t), \mathbf{u}(t), \mathbf{i}(t), \mathbf{p}, t)\mathrm{d}t \tag{1.7}$$

subject to:

$$\mathbf{f}(\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{u}(t), \mathbf{i}(t), \mathbf{p}, t) = 0, \qquad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{1.8}$$

$$\mathbf{g}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{i}(t), \mathbf{p}, t) \le 0, \qquad l = \overline{1, m_e + m_i} \tag{1.9}$$

$$\mathbf{u}_L \le \mathbf{u}(t) \le \mathbf{u}_U, \tag{1.10}$$

$$\mathbf{i}_L \le \mathbf{i}(t) \le \mathbf{i}_U, \tag{1.11}$$

$$\mathbf{p}_L \le \mathbf{p} \le \mathbf{p}_U, \tag{1.12}$$

where $\mathbf{x}(t) \in X \subseteq \mathrm{R}^{n_x}$ is the vector of state variables, $\mathbf{u}(t) \in U \subseteq \mathrm{R}^{n_u}$ is the vector of real valued control variables, $\mathbf{i}(t) \in I \in \mathrm{Z}^{n_i}$ is the vector of integer control variables,

$\mathbf{p} \in P \subseteq \mathrm{R}^{n_p}$ is the vector of time-independent parameters, t_f is the final time of the process, m_e, m_i represent the number of equality and inequality constraints, $\mathbf{f}$ is the set of ordinary differential equations describing the dynamics of the system (plus the corresponding initial conditions), $\mathbf{g}$ is the set of state constraints (path, pointwise and final time constraints), and $\mathbf{u}_L$, $\mathbf{i}_L$, $\mathbf{p}_L$, $\mathbf{u}_U$, $\mathbf{i}_U$, $\mathbf{p}_U$ correspond to the lower and upper bounds for the control variables and the time-independent parameters.

Parameter estimation problems

A particular type of dynamic optimization problem which needs a detailed explanation is *parameter estimation in dynamic systems*, i.e. calibration of models composed of differential equations. Building a dynamic biological model is an iterative process [20], usually represented as a cycle (see Figure 1.10). It starts with the definition of the purpose of the model and the selection of a model framework. Then, a mathematical structure is proposed with a set of non-measurable parameters. After that, these parameters are estimated in order to obtain quantitative predictions. Finally, the model is (in)validated with new experiments, obtaining feedback which can be subsequently used in a refinement process.

The parameter estimation step is key in this iterative model building process and can be formulated as a mathematical optimization problem subject to the dynamic constraints which describe the time-dependent behavior of the system. Most biological models are highly non-linear dynamical systems, resulting in challenging multi-modal problems which are very difficult to solve, as described in [218].

In computational systems biology, parameter estimation seeks to obtain a decision vector $\mathbf{p}$, that optimizes the cost function in order to obtain quantitative predictions which match a given set of experimental time-series data, satisfying other possible constraints. Finding the optimal values of this decision vector $\mathbf{p}$ can be represented as a NLP problem, where the objective is to minimize the cost function:

$$J = \sum_{\varepsilon=1}^{n_\varepsilon} \sum_{o=1}^{n_o^\varepsilon} \sum_{s=1}^{n_s^{\varepsilon,o}} (ym_s^{\varepsilon,o} - y_s^{\varepsilon,o}(\mathbf{p}))^T W (ym_s^{\varepsilon,o} - y_s^{\varepsilon,o}(\mathbf{p})) \tag{1.13}$$

where n_ε is the number of experiments, n_ε^ϵ is the number of the observables (state variables measured experimentally), $n_s^{\epsilon,o}$ corresponds with the number of the samples

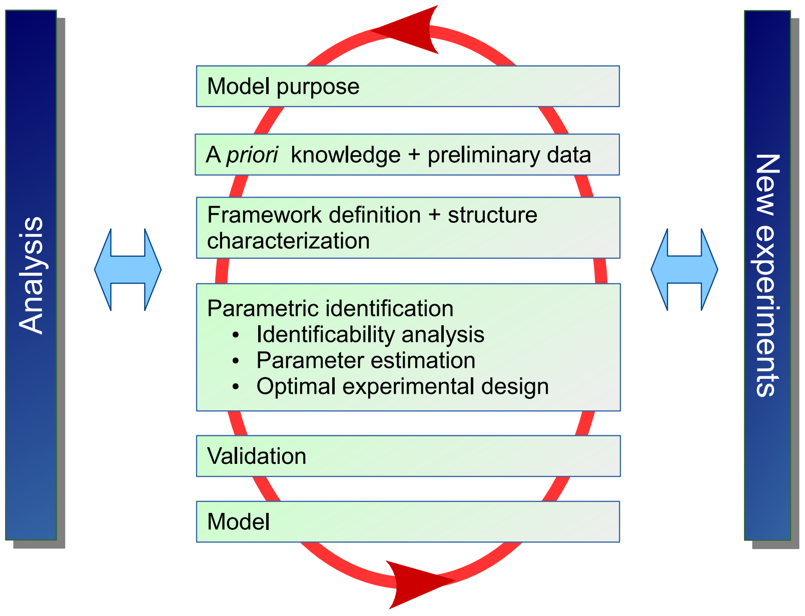

Analysis
Model purpose
A *priori* knowledge + preliminary data
Framework definition + structure characterization
Parametric identification
• Identificability analysis
• Parameter estimation
• Optimal experimental design
Validation
Model
New experiments

Figure 1.10: Model building loop. Adapted from [20].

per observable per experiment, $ym_s^{\varepsilon,o}$ are the measured data, $y_s^{\varepsilon,o}(\mathbf{p})$ are the model predictions, and W is a scaling matrix that balances the results of the observables. This optimization is also subject to the following constraints:

$$\dot{x} = f(x, \mathbf{p}, t) \tag{1.14}$$

$$x(t_o) = x_o \tag{1.15}$$

$$y = g(x, \mathbf{p}, t) \tag{1.16}$$

$$h_{eq}(x, y, \mathbf{p}) = 0 \tag{1.17}$$

$$h_{in}(x, y, \mathbf{p}) \leq 0 \tag{1.18}$$

$$\mathbf{p}^L \leq \mathbf{p} \leq \mathbf{p}^U \tag{1.19}$$

where x is the vector of state variables, x_o are the initial conditions, f is the non-linear dynamic problem with the differential-algebraic constraints, g corresponds with the observation function, h_{eq} and h_{in} are equality and inequality constraints, and p^L and p^U are lower and upper bounds for the decision vector $\mathbf{p}$. The equality dynamic constraints, Eqn. 1.14-1.15, are solved as an inner initial value problem for each decision vector. Note that Eqn. 1.17-1.18 constraints could be handled using differential-algebraic solvers and suitable penalty functions, as described in [21]. Upper and lower bounds for the parameters (Eqn. 1.19) could be handled by a reflection strategy [172] during the global phase.

All of the optimization problems described above may be applied to the field of systems biology. Some examples of theses applications are given in Table 1.1. A detailed discussion about optimization in computational systems biology and more examples can be found in [19].

Most of the optimization problems handled in this Thesis are considered within the parameter estimation procedure in dynamic models described by deterministic nonlinear ordinary differential equation models. The optimization problem obtained in the calibration of the model has also a NP-hard complexity, due to the non-linearity, non-convexity, multi-modality, and ill-conditioning in the topology described by its associate function. Thus, efficient methods are needed to solve this problem in practice.

1.2.2. Optimization methods

As shown above, most of the problems in computational systems biology are highly constrained and exhibit nonlinear dynamics. These properties often result in non-convexity and multi-modality. Therefore, there is a great demand for suitable optimization solvers for these problems.

Global optimization (GO) methods are robust alternatives to solve complex optimization problems, playing an increasingly important role in computational biology [84], bioinformatics [121] and systems biology [20]. They can be roughly classified into the following classes:

- *Deterministic GO methods.* These algorithms usually explore the entire search space, the solution retrieved being the global optimum. However, the associated computational effort to ensure global optimality might be extremely large, making them impractical.

Table 1.1: Examples of applications of optimization problems in computational systems biology. Adapted from [19].

Type	**Description**	**Examples**
LP	Linear objective and constrains	Metabolic flux balancing [163, 212]; genome scale models [173]; inference of regulatory networks [224, 225]
NLP	The objective function, or some of the constrains, are nonlinear	Parameter estimation and metabolic engineering [136]; analysis of energy metabolism [221]
MINLP	Nonlinear problem with both discrete and continuous decision variables	Develop of metabolic reaction networks and their regulatory architecture [89, 90]; inference of regulatory interactions using time-course DNA microarray expression data [204]
DO	Optimization with differential equations as constraints and possible time-dependent decision variables	Optimal experimental design [16]; discovery of biological network design strategies [2]; dynamic flux balance analysis [134]
MIDO	Optimization with differential equations as constraints and both discrete and continuous, and possible time-dependent, decision variables	Computational design of genetic circuits [49]
Parameter estimation in ODEs	Model calibration minimizing differences between predicted and experimental values	Parameter estimation using global and hybrid methods [18, 141, 181]; parameter estimation in stochastic methods [179]

- *Stochastic GO methods.* These methods do not guarantee convergence to the global optimum, but they usually provide near-global solutions in reasonable computation times. The most important groups of algorithms within this category are: adaptive stochastic methods, clustering methods and metaheuristics.
- *Hybrid GO methods.* These methods arise from a combination of two or more methodologies: a couple formed by a global method and a local search, the union of global optimization solver with a deterministic method, or a set of GO methods combined among them.

Among the different GO methods, we highlight metaheuristics, since the target of this Thesis is the design and implementation of novel approaches in this class.

Metaheuristics

A metaheuristic [59,131,196] is an iterative generation process that guides a subordinate heuristic by combining different concepts for exploration (global search) and exploitation (local search) of the search spaces. It uses learning strategies to structure information in order to find efficiently near-optimal solutions. Metaheuristics allow handling NP-hard optimization problems by providing good enough solutions in a reasonable computation time, because they do not need to explore the entire address space. Moreover, modern metaheuristics often use hybrid approaches where the global search includes also local searches to obtain a compromise between the diversification provided by the global optimization method, and the intensification obtained by the inclusion of a local method. However, there is no guarantee to find global optimal solutions or even bounded solutions.

Metaheuristics can be described as a search through the solution domain of the problem at hand. They are usually iterative procedures that move from a given solution to another solution in its neighbourhood. Thus, at each iteration, a evaluation moves towards solutions in the neighbourhood of the current solution, or in a suitably selected subset. According to various criteria a number of good moves are selected and implemented. However, the solutions implemented by metaheuristics do not necessarily improve. Some metaheuristics, such as tabu search or simulated annealing methods, usually implement one move at each iteration, while other

Algorithm 1: Generic metaheuristic.

```
1  Initialization;
2  repeat
3      Neighbourhood selection;
4      repeat
5          Candidate selection;
6          Move evaluation/Neighbourhood exploration;
7          Move implementation;
8          Solution evaluation, update search parameters;
9      until Stopping criteria or initiate new global search;
10 until Stopping criteria;
```

methods, like genetic algorithms, may generate several new moves at each iteration. Moves may marginally modify the solution (local search) or drastically inflect the search trajectory (global search). A generic metaheuristic procedure is illustrated in Algorithm 1.

Each metaheuristic has its own behaviour and characteristics. However, all of them share a number of fundamental components and perform operations that fall within a limited number of categories. Thus, metaheuristics can be classified attending to different criteria, not exclusive between them (see Figure 1.11):

- *Population-based search versus single-solution based search.* In population-based algorithms, such as differential evolution (DE) or scatter search (SS), there is a set of solutions stored in populations, which are handled and modified by the algorithm, while in single-solution based algorithms, such as in simulated annealing (SA), a single solution evolves during the search.
- *Nature inspired versus non-nature inspired.* Many algorithms have strategies or schemes inspired in elements of nature, such as genetic algorithms (GAs), ant colony optimization (ACO), particle swarm (PSO), or simulated annealing (SA). Other methods are not nature inspired, like differential evolution (DE), scatter search (SS), or tabu search (TS).
- *Iterative versus greedy.* Greedy algorithms begins from an empty solution and, at each step, a decision variable of the problem is assigned until a complete

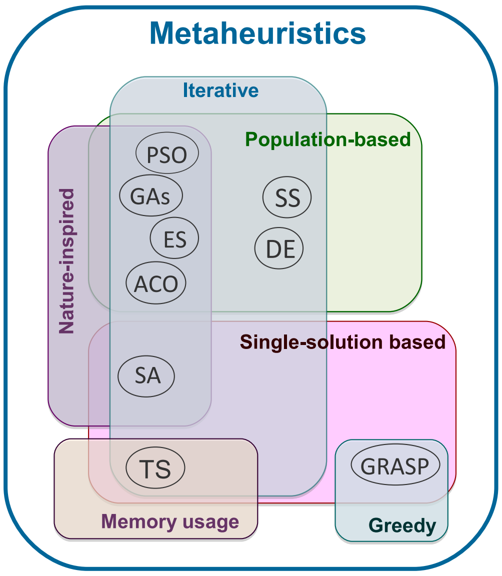


Figure 1.11: Classification of popular metaheuristics. Adapted from [4].

solution is reached. Iterative metaheuristics starts with one or more of solutions, evolving them in each iteration using some search operators. Most of the metaheuristics are iterative methods.

- *Memory usage versus memoryless methods.* Some algorithms use a memory to store information extracted during the search, such as memories in tabu search (TS) methods.

In the past decades, metaheuristics have received increasing attention, and a large amount of metaheuristics have arisen. Among the most well-known metaheuristics we can find the following [31]:

- Simulated Annealing (SA): inspired by the annealing technique used by the metallurgist. The objective function of the problem, similar to the energy of a material, is minimized, by introducing a fictitious temperature T, which is a simple controllable parameter of the algorithm. SA is a single-solution based algorithm that starts by generating an initial solution. At each iteration, a new solution is randomly selected in the neighborhood. This new solution is accepted depending on T and on the value of the objective function.

- Tabu search (TS): inspired by the human memory, and based on the premise that problem solving must incorporate *adaptive memory* and *responsive exploration*. Various types of memories are used to remember specific properties of the trajectory through the search space. A *tabu list*, that is, a short-term memory, records the last encountered solutions and forbids these solutions from being visited again. Additional intermediate-term memory can be introduced to influence the move towards promising areas of the search space (intensification), as well as long-term memory to encourage broader exploration of the search space (diversification).

- Greedy Randomized Adaptive Search Procedure (GRASP): a memory-less multi-start metaheuristic for combinatorial optimization problems. Each iteration of the GRASP algorithm consists of two steps: construction and local search. The construction step builds a feasible solution using a randomized greedy heuristic. In the second step, this solution is used as the initial solution of a local search procedure.

- Genetic algorithms (GAs): inspired on principles of natural selection and genetics. GAs encode the decision variables in sets of solutions (*chromosomes*), formed by different parts (*genes*) with some values (*alleles*). In GAs the population evolves during the algorithm through selection, recombination, mutation and replacement phases.

- Evolution strategy (ES): that also imitates the principles of natural evolution. It is based upon a population consisting of a single parent which produces, by means of normally distributed mutation, a single descendant. The selection operator then determines the fitter individual to become the parent of the next generation.

- Differential evolution (DE): one of the most popular algorithm for continuous global optimization problems. It is an evolutionary algorithm which uses vector differences for perturbing the vector population.

- Particle Swarm Optimization (PSO): inspired by social behavior of bird flocking or fish schooling. It starts initializing a population with random solutions, and then searches for optima by updating generations. However, PSO has no evolution operators, such as crossover and mutation in GAs. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles.

- Ant Colony Optimization (ACO): inspired by the social behaviour of some insects that present a sophisticated social structure. A set of agents (*ants*) search for good solutions finding the best path on a weighted graph.
- Scatter Search (SS): founded on the premise that systematic designs and methods for creating new solutions afford significant benefits beyond those derived from recourse to randomization.

Most of these metaheuristics were initially developed for combinatorial problems, that is, for problems where the decision variables are integer. Since optimization problems arising in computational systems biology are frequently continuous or mixed-integer, specific adaptations must to be done to classical metaheuristics in this context. In this Thesis, novel methods are proposed, based on the well known Differential Evolution and Scatter Search metaheuristics.

1.3. Parallel metaheuristics

Although the use of metaheuristics allows notably reducing the computational complexity of the search process, it still remains time consuming for many problems in multiple domains of application. High performance computing (HPC) represents an effective strategy to speed up the time-to-solution. Besides, as commented in Section 1.1, thanks to recent developments in technology, the use of parallel computing is becoming increasingly popular. As an example, current laptops and workstations are equipped with multicore processors. Additionally, the cost/performance ratio of HPC architectures is constantly decreasing, and, moreover, with the advent of the cloud computing effortless access to large number of distributed resources has become more feasible. Thus, the main target of this Thesis is the design and implementation of effective novel parallel metaheuristics to be executed in parallel and distributed computer systems.

Parallel computing traditionally follows from a decomposition of the total computational load and the distribution of the resulting tasks to available processors. The decomposition may concern the algorithm (*functional parallelism*) or the data (*data parallelism* or *domain decomposition*). Metaheuristics as algorithms may have limited data or functional parallelism. For example, the local search loop (lines 5-8) of the generic metaheuristic shown in Algorithm 1 displays strong data dependencies between successive iterations. However, the exploration of the solution space (external loop in Algorithm 1), if it is based on random restarts, can be functionally parallelized. Nevertheless, metaheuristics as problem solving methods offer other opportunities for parallel computing. A metaheuristic algorithm started from different initial solutions will almost certainly explore different regions of the solution space and return different solutions. The different regions of the solution space explored can then become a source of parallelism for metaheuristic methods.

1.3.1. Classification of parallel metaheuristics

We adopt here the taxonomy used in [46] to describe the different parallel strategies for metaheuristics. This classification employs three dimensions in the description of a parallel metaheuristic:

- *Search control cardinality*: indicates how the global problem-solving process is controlled. It specifies whether the global search is controlled by a single process or by several processes that may collaborate or not. The two alternatives are *1-control* (1C) or *p-control* (pC).
- *Search control communications*: indicates how information is exchanged among processes and the quality of this information. It deals with the communication protocols used for information exchange. To reflect adequately how information is exchanged, four alternatives are identified for this dimension:
    - *Rigid* (RS): when synchronous communication protocols are used and all concerned processes have to stop and engage in information exchange at moments (number of iterations, time intervals, etc.) externally determined.

  - *Knowledge Synchronization* (KS): when synchronous communication protocols are used but an increased level of communication permits building and exchanging knowledge.
  - *Collegical* (C): when asynchronous communication protocols are used, that is, each process is in charge of its own search, as well as establishing communications with other processes.
  - *Knowledge Collegical* (KC): when asynchronous communication protocols are used and the quality of the information allows for extracting new information to guide the process.

- *Search differentiation*: indicates the variety of solution methods involved in the search for solutions. It specifies whether the search processes start from the same or different solutions and if they make use of the same or different search strategies. The four cases considered are:
  - Same initial point/Population, Same search Strategy (SPSS)
  - Same initial point/Population, Different search Strategies (SPDS)
  - Multiple initial points/Populations, Same search Strategy (MPSS)
  - Multiple initial points/Populations, Different search Strategies (MPDS)

According to this classification, the parallel metaheuristics proposed in this Thesis can be classified as:

- *Search control cardinality*: both 1C (case of saCeSS algorithm presented in Chapter 3 and saCeSS2 presented in Chapter 4) or pC (case of asynPDE algorithm presented in Chapter 2 or aCeSS, that was a preliminary version of saCeSS, and is also commented in Chapter 3).
- *Search control communications*: either C or KC, because all of them use asynchronous communication protocols and in some of them the exchanged information allows guiding the search process (case of saCeSS presented in Chapter 3 and saCeSS2 presented in Chapter 4).
- *Search differentiation*: MPDS, because in the approaches proposed all the processes start from different solutions and perform different search strategies.

Nevertheless, the implementation of all the algorithms proposed has been done to also allow for an *homogeneous* configuration, that is, all of them can be also executed as MPSS if desired.

A simpler classification of parallel metaheuristics, that can be also applied to the novel metaheuristics proposed in this Thesis is presented in [45]:

- Type 1. This source of parallelism is usually found within an iteration of the heuristic method. The limited parallelism of a move evaluation is exploited, or moves are evaluated in parallel. This strategy is rather straightforward and aims solely to speed up computations, without any attempt at achieving a better exploration or higher quality solutions.
- Type 2. This approach obtains parallelism by partitioning the set of decision variables. The partitioning reduces the size of the solution space, but it needs to be repeated to allow the exploration of the complete solution space. Obviously, the set of visited solutions using this parallel implementation is different from that of the sequential implementation of the same heuristic method.
- Type 3. Parallelism is obtained from multiple concurrent explorations of the solution space.

Parallel metaheuristics proposed in this Thesis attempt to combine parallelism in the previous three groups: thanks to fine-grained parallelizations that perform evaluations in parallel some of them are type 1; at the same time, when the population is partitioned between different processes, they can be classified as type 2; and finally, we attempt to explore the solution space concurrently, even using different strategies, so they should be also considered of type 3.

1.3.2. Metaheuristics parallelization goals

In general, the parallelization of metaheuristics pursues one or more of the following goals:

- Speed up the search. One of the main goals of the parallelization of any algorithm is to reduce the execution time. In the case of metaheuristics, this means reducing the search time. This is specially important for some problems where there are hard requirements on the search time, such as dynamic optimization problems and time critical control problems.

- Improve the quality of the obtained solutions. Algorithm level solutions, based on parallel cooperative processes, allow to improve the quality of the search, because the exchange of information between cooperative metaheuristics will alter their behavior in terms of searching in the landscape associated with the problem.
- Improve the robustness. A parallel metaheuristic may be more robust in terms of solving, in an effective manner, different optimization problems and different instances of a given problem. Robustness may also be measured in terms of the sensitivity of the metaheuristic to its parameters.
- Solve large-scale problems. Parallel metaheuristics allow solving large-scale instances of complex optimization problems. One of the challenges of the parallel solutions is to solve very large instances that cannot be solved with individual searches or without parallel resources.

The solutions proposed in this Thesis aim to fulfill all these goals. The main target will be to propose novel parallel metaheuristics that outperform the state-of-the-art techniques for the problems considered. Thus, the evaluation of the proposals is a key step in our work. However, it is difficult to make fair comparisons between metaheuristics, because different conclusions can be inferred from the same results depending on the metrics we use and how they are applied.

1.3.3. Performance evaluation

Most of the metaheuristics, whether parallel or serial ones, are evaluated *empirically* in an *ad hoc* manner, due to the difficulty in developing theoretical analysis [4]. An experimental analysis usually consists in applying the proposed algorithms to a collection of problem instances and comparatively report the observed solution quality and execution time.

In deterministic optimization methods, the efficiency in terms of search time is the main factor to evaluate the performance of the algorithms, since they guarantee the global optimality of solutions. However, to evaluate metaheuristic methods, other measures have to be considered. Because of the stochastic nature of metaheuristics, a number of independent experiments need to be conducted to gain sufficient experimental data. Thus, the performance measures for these methods are based on some kind of statistics.

This subsection describes the guidelines followed in this Thesis to evaluate, and also to compare, the parallel metaheuristics proposed in a rigorous way.

Vertical versus horizontal views

There exist two different approaches for collecting data of different runs [87]:

- Vertical view: a vertical approach assesses the performance for a predefined effort. The effort may be predefined as a target execution time or as a fix number of evaluations. Fixing a predefined effort can be pictured as drawing a vertical line on the convergence graphs (see Figure 1.12).
- Horizontal view: an horizontal approach assesses the performance by measuring the time needed to reach a given target value. Fixing a target function value can be seen as drawing an horizontal line in the convergence graphs (see Figure 1.12).

For benchmarking algorithms the horizontal view is preferred to the vertical one, since it gives quantitative and interpretable data: the horizontal view measures the time needed to reach a given target function value and allows deriving conclusions such as Algorithm A is X times faster than Algorithm B in solving this problem. In the vertical view, there is no interpretable meaning to the fact that Algorithm A reaches a fitness value that is X times smaller than the one reached by Algorithm B, since there is no a priori evidence how much more difficult it is to reach a fitness value that is X times smaller (as demonstrated in Figure 1.12).

The main goal of this Thesis is to improve the horizontal approach, however, the vertical approach also benefits from the proposed solutions, which is interesting for

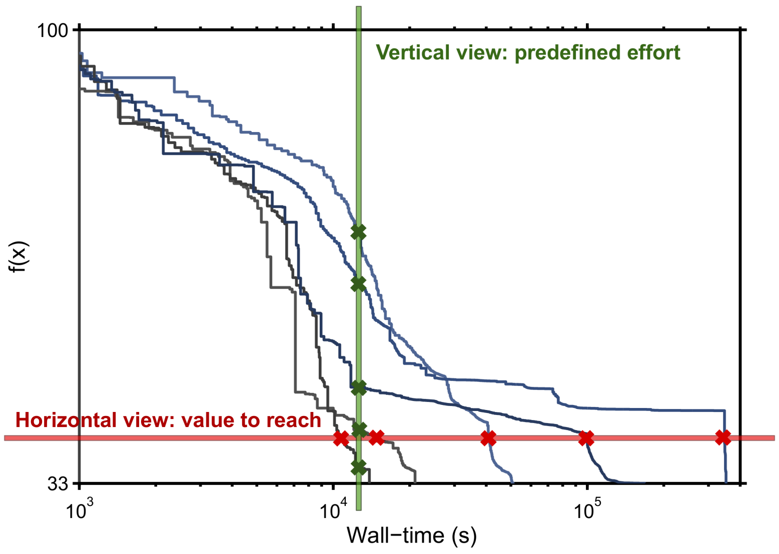


Figure 1.12: Vertical and horizontal views illustrated in a convergence graph. Adapted from [87].

many real world applications where the total number of evaluations is limited.

Speedup

There are different metrics to measure the performance of parallel algorithms. Among them, the *speedup* is the most popular one. This metric calculates the ratio between sequential and parallel execution times. Thus, the definition of execution time must be faced. In a single core, a common performance metric is the *CPU time* to solve the problem, that is, the time that the processor spends executing algorithm instructions, excluding system overhead activities. However, in the parallel case, execution time can not be considered either the sum of the CPU times on each core, or the largest among them. Since the goal of parallelism is the reduction of the real time, our choice for measuring the performance of the parallel code is the *wall-clock time* to solve the problem, that is, the time between the starting and finishing of the entire algorithm.

The speedup compares the sequential time against the parallel time to solve a problem. If T_n is the execution time for a parallel algorithm using n cores, the speedup is the ratio between the faster execution time on a single core, T_1, and T_n:

$$S_n = \frac{T_1}{T_n}$$

Unfortunately, for stochastic algorithms we cannot use this metric directly. The speedup should be instead calculated using the *mean* execution times. Moreover, another issue with this measure is that researchers do not agree on the meaning of T_1 and T_m, and there exists different definitions of speedup depending on the meaning of these values [4]:

- *Strong speedup*: compares the parallel run time against the best-so-far sequential algorithm. This is the most accurate definition of speedup, but due to the difficulty of finding the current most efficient algorithm, it is not a practical one.
- *Weak speedup*: compares the parallel algorithm developed by a researcher against his/her own sequential version. This is the definition of speedup used in this Thesis.

Additionally, two different sequential algorithms can be considered to calculate the speedup. We can compare the execution time of the parallel algorithm against the canonical sequential version of the algorithm (*speedup versus panmixia*) or we can compare the execution time of the parallel algorithm against the same parallel algorithm running on one core (*orthodox speedup*). In this Thesis, we always use the former, that is, a speedup versus panmixia. However, it should be noted that in this case we are comparing two clearly different algorithms, and thus, superlinear speedups may arise when the parallel algorithm modifies the systemic properties of the original method and outperforms the sequential algorithm.

Graphical data representation

Some visualization tools to analyze the data have to be used to complement the numerical results presented in tables. Indeed, graphical representation of the data allows a better understanding of the performance assessment of the obtained results.

Boxplots illustrate the distribution of the results through their five-number summaries: the smallest value, lower quartile (Q1), median (Q2), upper quartile (Q3), and largest value. They are useful in detecting outliers and indicating the dispersion and the skewness of the output data without any assumptions on the statistical distribution of the data. Violinplots, in turn, show the probability density of the data at different values. While the boxplots only show summary statistics, the violinplots show the full data distribution. The difference is particularly useful when the data distribution is multimodal (more than one peak). The violin plots clearly shows the presence of peaks, their position and relative amplitude. Violinplots are a good alternative to employ a serie of histograms. In this Thesis, a combination of boxplots and violinplots (see Figure 1.13) are used in order to incorporate the goals of both representations.

To clearly illustrate the goal of a proposed metaheuristic against other approaches convergence curves are frequently used. Convergence curves (see Figure 1.14) represent the logarithm of the objective function value against the execution time. Though one can argue about the convenience of representing the convergence curves for the best profits, or even artificial convergence curves obtained by plotting the best solution found by any of the n parallel processes at every time instant [219], in this Thesis we prefer to show the converge curves for those experiments that fall in the median value of the results distribution, since those are real convergence curves (that is, correspond to one of the experimental tests) and we think they are more realistic than the ones that depicted the best profit. The region between the lower and upper bounds of the m runs performed for each experiment can also be shown to better illustrate the dispersion of the results (see Figure 1.14(b)).

Statistical analysis

Most of the reported results in this Thesis try to prove that a novel proposed metaheuristic outperforms previous attempts. In this case, the use of descriptive statistics, such as the sample mean and the standard deviation, is not sufficient. The comparison between two average values may be different from the comparison between two distributions. Thus, statistical methods should be used wherever possible. The statistical test are performed to estimate the confidence of the results to be scientifically valid.

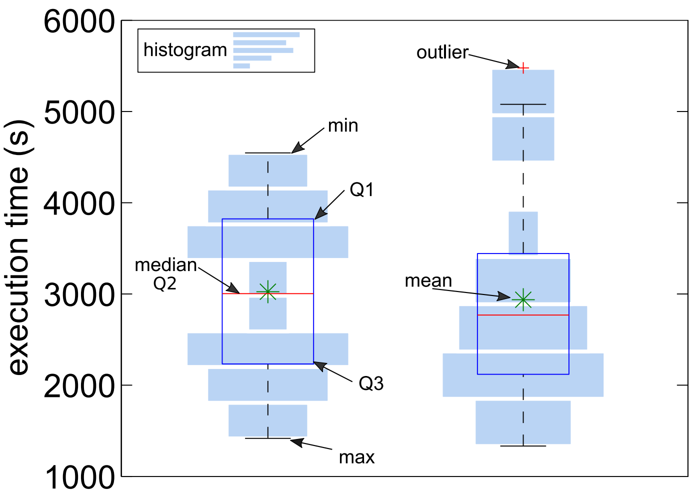
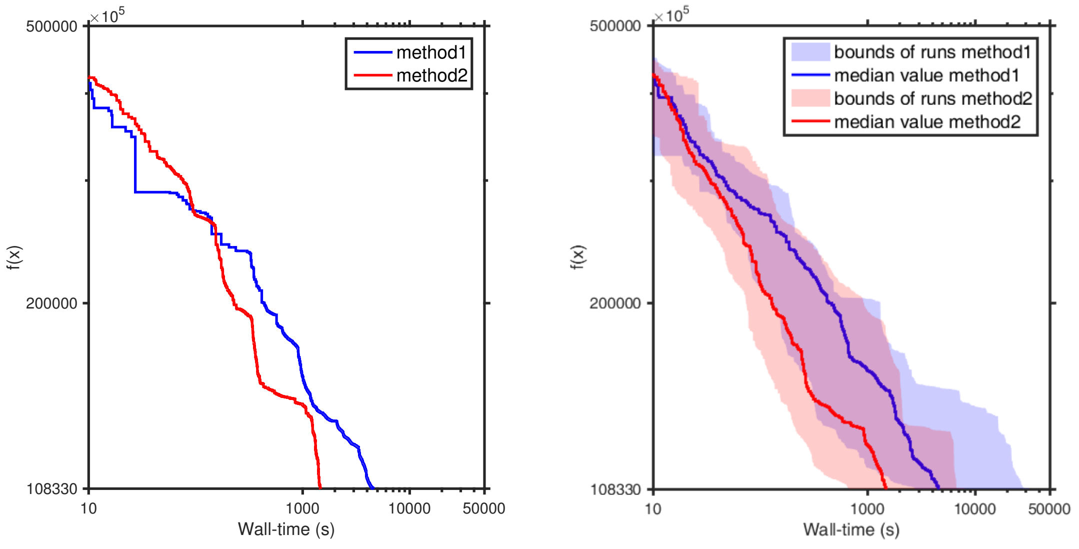


Figure 1.13: Example of hybrid violin/boxplot.



(a) Convergence curves

(b) Convergence curves including lower and upper bounds

Figure 1.14: Example of convergence curves.

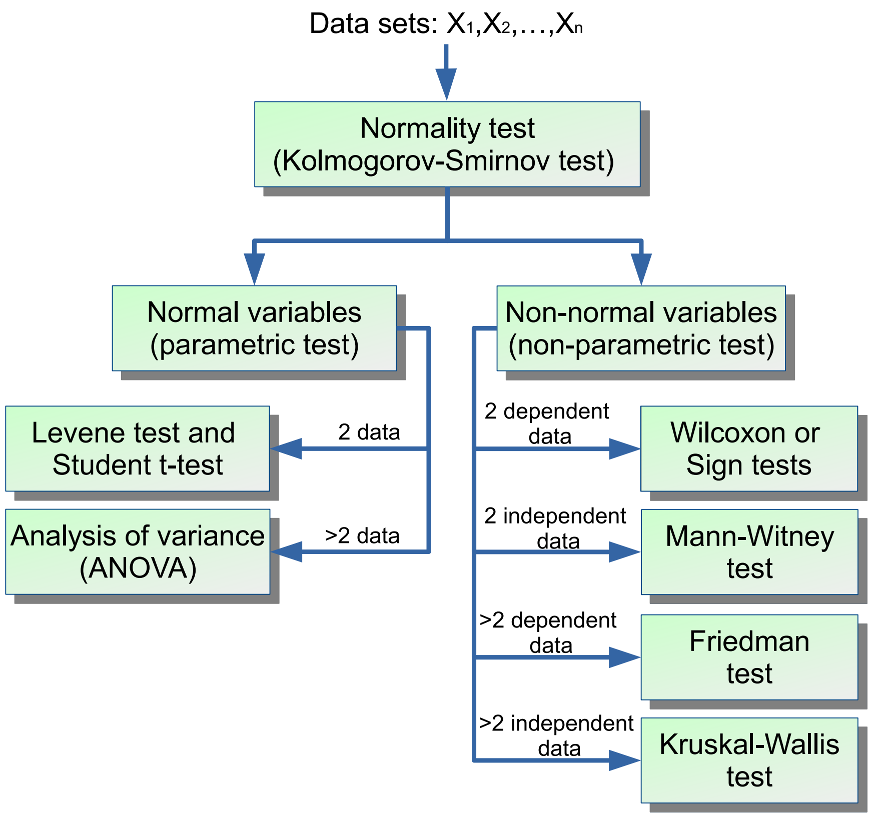


Figure 1.15: Statistical methods. Adapted from [4].

.

Several statistical methods and the conditions to apply them are shown in Figure 1.15. The selection of a given statistical test is driven according to the characteristics of the data [4]. The first step is to decide between non-parametric and parametric test. In theory, when the data set is non-normally distributed and the number of experiments is below 30 we should use non-parametric methods. That is the case in the experiments performed in this Thesis.

Among non-parametric methods, in this Thesis the Wilcoxon signed-rank test has been used when comparing two metaheuristics, and the Kruskal-Wallis test has been used when comparing more than two algorithms. The Wilcoxon signed-rank test assumes that there is information in the magnitudes and signs of the differences between paired observations. This test essentially calculates the difference between each set of pairs and analyzes these differences. Then, the Wilcoxon rank-sum test can be used to test the null hypothesis that two populations have the same continuous distribution. When more than two samples are compared, the Kruskal-Wallis test is used, that is also based on ranked data.

Chapter 2

Enhanced parallel Differential Evolution

Differential Evolution (DE) [194] is one of the most popular heuristics for global optimization, and it has been successfully used in many different areas [42,48,172]. In particular, DE remains as a widely used method for parametric identification of complex models [47, 175, 241]. However, in most realistic applications, this population-based method requires a very large number of evaluations (and therefore, large computation time) to obtain an acceptable result. Thus, in order to improve the runtime of the classic DE algorithm, we have explored two different strategies. First, including a selected local search and other algorithmic improvements in order to enhance the classic DE through intensification, drastically reducing the number of evaluations required. Second, exploiting parallelism at different levels, so as to reduce the computational time needed.

The organization of this chapter is as follows. Section 2.1 presents a brief overview of the DE algorithm, while Section 2.2 covers the related work. Section 2.3 describes the asynchronous strategy proposed to parallelize an island-based DE algorithm. Section 2.4 describes the new heuristics for parameter estimation problems that have been added to improve local search. The performance of the proposed method is evaluated in Section 2.5, demonstrating its good efficiency and scalability. Finally, Section 2.6 summarizes the main conclusions of this chapter.

2.1. Differential Evolution algorithm

Differential Evolution (DE) is an iterative mutation algorithm where vector differences are used to create new candidate solutions. Algorithm 2 shows a simple pseudocode for the classic sequential DE.

Starting from an initial population matrix composed of NP D-dimensional solution vectors (individuals), DE attempts to achieve the optimal solution iteratively through changes in its vectors. For each iteration, new individuals are generated in the population matrix through mutation operations performed among individuals of the matrix. These mutation operations depend on the mutation factor (F), which is used in the creation of new solutions in different ways depending of the selected mutation scheme. There are different mutation strategies (MSt) to generate new individuals, this chapter uses some of them:

- `DE/best/1`:
$$Ind_k \leftarrow bestP_k + F \cdot (b_k - c_k) \tag{2.1}$$
- `DE/best/2`:
$$Ind_k \leftarrow bestP_k + F \cdot (b_k - c_k) + F \cdot (d_k - e_k) \tag{2.2}$$
- `DE/rand/1`:
$$Ind_k \leftarrow a_k + F \cdot (b_k - c_k) \tag{2.3}$$
- `DE/rand/2`:
$$Ind_k \leftarrow a_k + F \cdot (b_k - c_k) + F \cdot (d_k - e_k) \tag{2.4}$$

where $\overrightarrow{a}, \overrightarrow{b}, \overrightarrow{c}, \overrightarrow{d}, \overrightarrow{e} \in P$ are solution vectors randomly selected, $\overrightarrow{bestP_k} \in P$ is the current best solution of the population, and $\overrightarrow{Ind} = (Ind_1, \ldots, Ind_D)$ is the new candidate solution created in the mutation process.

These mutation operations are applied in specific positions k of the old solution vector of the matrix. These positions are determined through the crossover constant (CR): if random generated value is less than CR, the mutation strategy is applied in the position k of the current vector. Thus, candidate solutions have one part of the old solutions that are intended to replace, and on the other hand, they have values obtained from the mutation process.

Algorithm 2: Differential Evolution algorithm - `seqDE`.

```
input  : A population matrix $P$ with size $D$ x $NP$
output: A matrix $P$ whose individuals were optimized
repeat
    for each element in $P$ do
        $\overrightarrow{a}, \overrightarrow{b}, \overrightarrow{c} \leftarrow$ different random individuals from P matrix
        for $k \leftarrow 1$ to $D$ do
            if random point is less than $CR$ then
                $\overrightarrow{Ind}(k) \leftarrow \overrightarrow{a}(k) + F(\overrightarrow{b}(k)$ - $\overrightarrow{c}(k))$;
            end
        end
        if Evaluation($\overrightarrow{Ind}$) is better than Evaluation($\overrightarrow{P}(x)$) then
            $\overrightarrow{P}(x) = \overrightarrow{Ind}$
        end
    end
until Stop conditions;
```

Finally, only when the fitness value of the new candidate solution is better than the current one, the new individual is included in the population matrix.

A population matrix with optimized individuals is obtained as output of the algorithm. The best of these individuals are selected as solution close to optimal for the objective function of the model.

2.2. Related work

Many researches have tried to improve DE by proposing modifications to the original algorithm. Interesting reviews can be found in [150] and more recently in [48]. In several cases, the original DE algorithm was improved with additional algorithmic components exploiting certain aspects of a given class of problems. In [231] a modified DE approach using generation-varying control parameters is proposed to improve the search performance preventing a premature convergence to local minima. A hybrid algorithm using DE as an evolutionary framework and a crossover-based local search was proposed in [152, 153]. A DE with Scale Factor Local Search was introduced in [205] and extended in [151] for self-adaptive DE schemes. The use of a tabu list in the DE has also been applied in recent works [120, 191, 193].

Several studies have considered parallel versions of the algorithm. A distributed adaptive DE version was proposed in [236]. This algorithm was based on the island-mode paradigm with a random communication topology for individuals exchange. Another parallel approach was proposed in [197], based on the distribution of the population data among different processors (slaves), which communicate through data migrations, all of them managed by a central processor (master). The latter was also responsible of checking the stopping criteria. The algorithm was implemented in PVM (Parallel Virtual Machine) [78] with presumably synchronous communication, resulting in low speed-up results.

A simple approach to asynchronous parallelization was proposed in [154], consisting of a master-slave architecture with several independent processes, where the communications were not established directly but through the filesystem. The master process was in charge of selecting the best individuals. When the stopping condition was satisfied, the slaves were stopped. Another asynchronous proposal based on a master-slave approach is the parallel metaheuristic based on DE and simulated annealing proposed in [155]. The master asynchronously assigns different tasks to the slaves, thus, allowing for simultaneous evaluation of several trial solutions. The proposal is implemented in MATLAB, using PVM for the communications between master and slaves.

A parallel DE with asynchronous communications and an island-model scheme is presented in [101]. Its termination criteria was controlled by a master node and was implemented with POSIX (Portable Operating System Interface) threads. A heterogeneous local configuration was used, where each parallel processor had a different mutation strategy. The results indicated an improvement in the reliability and the performance of the algorithm with a configuration of five islands, compared to the time spent by five sequential versions for the same optimization problems. Another asynchronous distributed DE was presented in [13]. In this case, the algorithm also exploits an island-model with asynchronous communication. The topology was an unidirectional ring, where the individuals exchanged were selected randomly.

Several other works studied improvements to island-model schemes. In [182], a complete study about the impact on the performance of different communication topologies of the islands was presented. These authors used a synchronous parallel DE on a set of standard benchmarks with different topologies, concluding that ring topology was the best option. In [226], subpopulations were grouped into two families. The first family uses a ring topology and a best-random like migration strategy to evolve its subpopulations. In the second one each subpopulation independently evolves with a population size reduction strategy. The solutions generated by the second family are moved into the first family.

Several studies suggest that randomization of the control parameters can be a propitious mechanism for enhancing the DE performance [33]. Different randomization schemes have been proposed to develop self-adaptive DE frameworks and investigate the effect of changing control parameters in distributed DE [228–230, 238]. Two mechanisms to avoid the loss of diversity when the size of the population is small are described in [227]. The first one was based on shuffling: the individuals from a specific subpopulation were randomly reorganized. The second one, an update mechanism, changed and adapted scaling factors for each subpopulations. The results indicated that these techniques obtained a very significant performance when the dimensionality of the functions grew. An improved strategy which entails four different scale factors updating schemes, associated to the binomial crossover in a distributed DE structure, is presented in [229]. With the exception of one scheme in which equally spaced scale factors are considered, in all the others the scale factors are randomly initialized for each subpopulation. Although proper choice of a scale factor scheme appears to be dependent on the distributed structure, each of these simple schemes proposed has proven to significantly improve upon the single-scale factor distributed DE algorithms.

Other parallelization strategies have appeared with the emergence of new hardware and software technologies. This is the case of [240], where a DE improved through local Pattern Search method was parallelized through CPU-GPU heterogeneous computing, using a cellular model scheme. Also, a parallel DE based on GPUs is explored in [222], which employs self-adapting control parameters and generalized opposition-based learning (GOBL) to improve the quality of candidate solutions. In [15] a multiagent framework was proposed to create a distributed cooperative structure based on agents. This scheme was implemented in Java, defining a communication API (Application Programming Interface) to send information to the different agents of the environment.

In this chapter, our aim was (i) to improve the DE algorithm incorporating several algorithmic steps which exploit the structure of parameter estimation problems, and (ii) develop a parallel version following a cooperative asynchronous strategy. The overall objective was to obtain a high performance implementation that achieves a good trade-off between exploration (diversification or global search) and exploitation (intensification or local search), which is at the core of modern metaheuristics [215].

2.3. Improving global search through an asynchronous parallel cooperative scheme

To achieve an efficient parallelization of metaheuristics is usually a complex task, since the search for new solutions depends on previous iterations of the algorithm, which not only complicates the parallelization itself but also limits the achievable speedup. Different strategies can be used to address this problem: attempting to find parallelism in the sequential algorithms, preserving its behavior; finding parallel variants of the sequential algorithms, slightly varying their behavior to obtain a more easily parallelizable algorithm; or developing fully decoupled algorithms, where each process executes its part without communication with other processes, at the expense of reducing its effectiveness. The solution explored in this Thesis pursues the development of an efficient parallel variant of the serial DE, focussed on both the acceleration of the computation by performing separate evaluations in parallel, and the convergence improvement through the stimulation of the diversification in the search and the cooperation between the parallel threads.

The parallel algorithm proposed is based on the island model approach [5]. The population matrix is divided into subpopulations (*islands*) where the algorithm is executed isolated. Phases such as selection, recombination and mutation are performed only within each island, which implies absence of collaboration among processes. Sparse individual exchanges are performed among islands to introduce diversity into the subpopulations, preventing search from getting stuck in local optima. After

m iterations, a migration phase links the different populations: selected individuals from each island are communicated to another island. Both the migration operation and the checking of the termination criteria imply an exchange of communications among processes.

The simplest implementation of the parallel island DE is a synchronous algorithm. The drawback of the synchronous algorithm is that processors are idle during a significant amount of time, while they are waiting for each other during the migration steps. Replacing synchronous communications with asynchronous ones, each process will send the information to a memory buffer associated with the remote process, enabling the reception of the message later on (whenever that process is ready to receive it), thus, avoiding idle periods. However, in such asynchronous version, when a migration is planned, each process would need to wait for the reception of its required new data. This could stall processes and cause idle periods.

The algorithm proposed in this Thesis avoids this by implementing a variation of the classic parallel island DE. The pseudocode for the proposed solution (`asynPDE`) is shown in Algorithm 3. Each process receives an island population. For each iteration of the main loop, mutation and crossover operations are performed within each island, in the same way as in the serial implementation. Every m iterations, a migration phase is performed to link the evaluations in different islands. Whenever a process reaches the migration phase, it sends a set of individuals to the selected remote process using an asynchronous communication (`ISend()` function in Algorithm 3). Then, the process in the migration phase checks if the message with the new individuals of a remote process has already arrived to its memory buffer (`IRecv()` function in Algorithm 3). However, if the new solutions have not yet arrived, the process proceeds with the next evaluation. After each evaluation the process searches for the reception of data missed in previous migrations (`Test()` function in Algorithm 3), however avoiding stalls if the messages have not arrived yet. To deal with the migration data received asynchronously, a dynamic data list is created. In each migration phase a new node is attached to the list, to be filled with the expected data. Once the data of a node is used, the node is removed from the list.

In addition to the migration step, the checking of the stopping criteria may also involve communications between processes. Stopping criteria are needed to termi-

Algorithm 3: Asynchronous island-based parallel Differential Evolution (`asynPDE`).

```
input  : A population matrix $P$ with size $D$ x $NP$
output: A matrix $P$ whose individuals were optimized
1  Plocal ← scatter population $P$ into $N$ processors
2  iter = 0;
3  repeat
4      for each element in Plocal do
5          Crossover, mutation and evaluation operations
6      end
7      ! migration phase
8      if iter%m==0 then
9          migrationSet ← selected individuals from Plocal
10         ! asynchronous send of migrationSet to remote destination
11         ISend(migrationSet, remoteDestination);
12         ! asynchronous reception of migrationSet in the receptionSet
13         IRecv(receptionSet, remoteOrigin); ! non-blocking operation that allows for
             execution progress if no message has arrived yet
14     end
15     while pending migration do
16         ! check for pending messages of previous migrations
17         Test(receptionSet, isComplete); ! non-blocking operation that allows for
             execution progress
18         if isComplete then
19             ! Insert received individuals into Plocal subpopulation
20             Plocal (selected individuals) ← receptionSet
21         else
22             break;
23         end
24     end
25     iter++;
26 until Stopping condition;
27 Gather all subpopulation into matrix $P$
```

nate the execution of the algorithms. They can be as simple as using a maximum number of evaluations, which do not imply exchange of communications. However, other criteria, that allow to react adaptively to the state of the optimization progress, need communications between processes. Asynchronous MPI communications are also used in the proposed algorithm for those communications, so that processes may continue running independently. Each parallel process opens a buffer where it expects to receive a termination message. This buffer is checked every iteration of the algorithm. Thus, the control of the stopping criteria of the global search is distributed among all the processes: when a stopping condition is fulfilled in a process, this condition is communicated to the rest of processes, then all of them can stop the algorithm almost at the same time.

Finally, note that the new parallel algorithm does not implement straightforwardly the serial one. As demonstrated in Section 2.5, it always performs better in terms of execution time and scalability even if, often, it requires higher number of evaluations to achieve convergence.

2.4. Improving local search in DE algorithms

In some real applications, such as parameter estimation in dynamic models, the performance of the classic sequential DE is not acceptable due to the large number of objective function evaluations needed. As a result, typical runtimes for realistic problems are in the range from hours to days. In order to improve the computational effort required by the DE algorithm running in each of the processors in the parallel version, three enhancements that exploit the special structure of these parameter estimation problems have been included.

2.4.1. Logarithmic space

The classic DE algorithm begins by generating an initial set P of individuals. Typically this generation is performed in a uniform space, where the range is divided into n sub-ranges of equal size and values are randomly chosen from selected sub-ranges. However, in other metaheuristics it has been found that, when variables may have values in a huge range, this uniform distribution for selecting diverse solutions will not generate many trial points with good value [62]. In contrast, a logarithmic distribution will initialize the algorithm with high quality members in the initial population, ensuring a faster convergence. Thus, the search is proposed to be performed in a logarithmic space, which results in a more suitable exploration of the space of parameters when they are positive and potentially span through several orders of magnitude.

2.4.2. Local solver

A local method is introduced to achieve a fast local convergence, therefore, reducing the number of objective function evaluations required when an horizontal view is assessed. The local search moves from solution to solution in the space of candidate solutions by applying local changes, until a solution considered optimal is found or a time bound is elapsed. NL2SOL [54], a method for solving non-linear least-squares problems, has demonstrated to be particularly effective for parameter estimation problems [62]. In the implementation of DE proposed in this chapter, NL2SOL is called every L iterations of the DE algorithm.

2.4.3. Tabu list

One drawback of the previous local search is that it tends to become stuck in suboptimal regions or on plateaus where many solutions are equally fit. As a means to avoid this problem, the concept of *tabu search* is introduced in the algorithm. Tabu search enhances the performance of local methods by avoiding revisits to the same place during the search. This is achieved using memory structures that describe the visited solutions. If the vicinity of a potential solution has been previously visited within a certain short-term period it is marked as *tabu*, so that the algorithm does not consider that possibility repeatedly. This technique improves the diversity among members of the population, and consequently contributes to the computational efficiency of the algorithm.

Algorithm 4 shows the pseudocode for the local search and tabu list included at the end of each external iteration of the DE algorithm. The local solver condition is met each L external iterations of the DE algorithm, and the evaluations performed during the local search stage count in the total number of evaluations of the DE algorithm.

Although all these three enhancements have significantly contributed to acceler-

Algorithm 4: Local Search and Tabu List.

```
TabuList ← set of visited points in the local solver ;
! local solver condition
if iter%L==0 then
    Sort Plocal population;
    $\overrightarrow{Point} = \overrightarrow{Plocal}(0)$;
    for $i \leftarrow 0$ to number of individuals of Plocal do
        ! Calculate all distances among the point Plocal(i) and all points of the
          Tabu List
        distanceSet = $distance_{Euclidean}(\overrightarrow{Plocal(i)}\ , TabuList)$;
        if $\forall\ d \in distanceSet > min\_distance$ then
            $\overrightarrow{Point} = \overrightarrow{Plocal}(i)$;
            break;
        end
    end
    Insert $\overrightarrow{Point}$ in the TabuList;
    $\overrightarrow{newPoint}$ = Run_Local_Search($\overrightarrow{Point}$);
    if $\overrightarrow{newPoint}$ is better than $\overrightarrow{Point}$ then
        Replace the worst point in Plocal with $\overrightarrow{newPoint}$;
    end
end
```

ate the solution of our real systems biology applications, the use of a local solver (effectively creating a hybrid method) has proved to be particularly useful. It is worth mentioning that hybrid methods have a long tradition in numerical methods in general, and numerical optimization in particular (e.g. Powell's dogleg method [171] is a well known classic example). In evolutionary computation, memetic algorithms [148] use a hybrid approach to combine global with local search methods. Hyper-heuristics make use of an even higher level of generality, with the objective of choosing the right heuristic for a given problem [36]. Here we have chosen a parsimonious hybrid design, combining a classic DE scheme with an specialized local search.

Figure 2.1 graphically illustrates the asynchronous parallel implementation of a DE extended with improved heuristics. Note that different processes are executing a DE in different stages, and cooperation between them is performed in an asynchronous fashion avoiding stalls if any of the processes is involved in a time consuming phase, such as the execution of the local solver (see process ID 3 in the

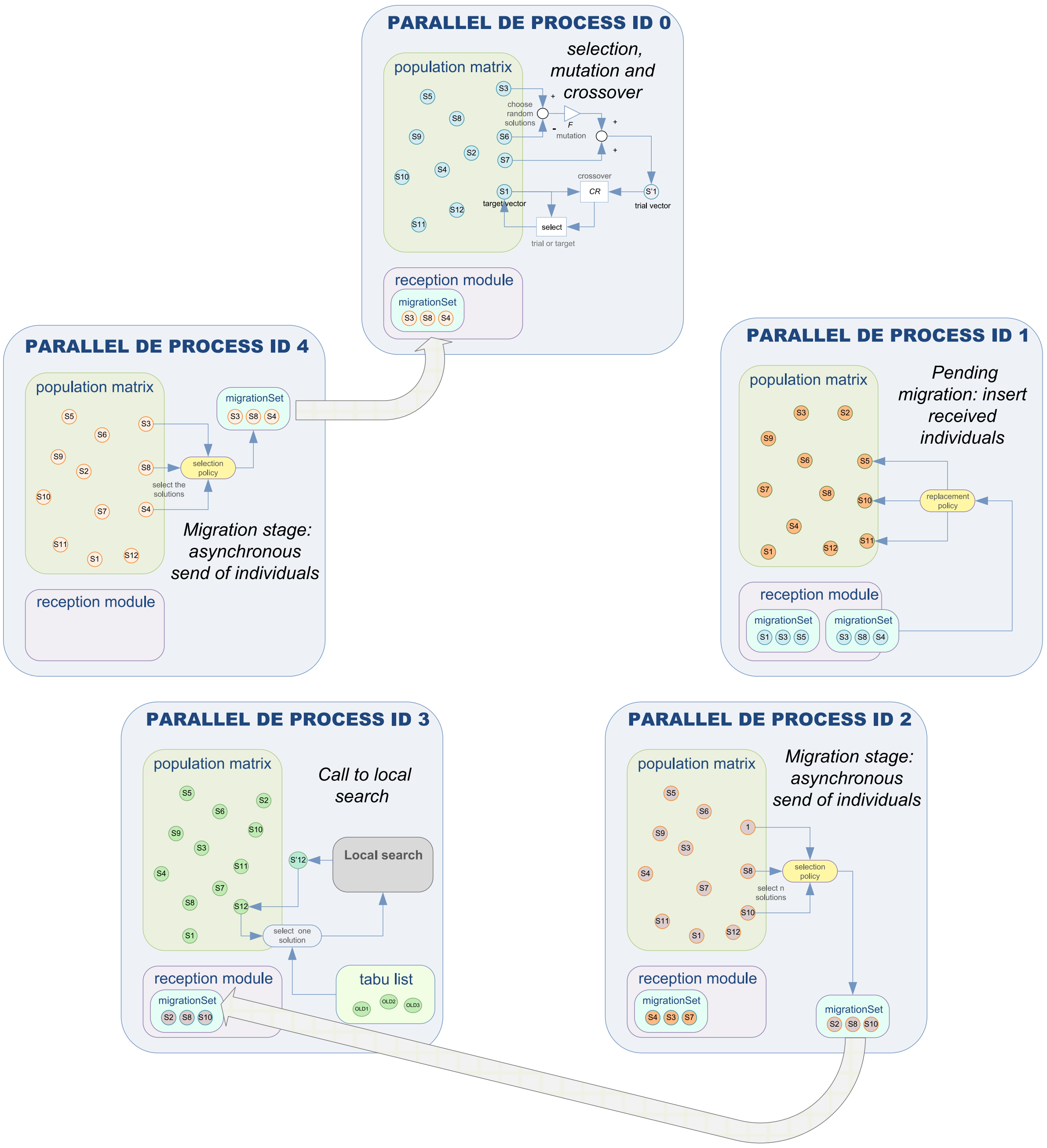


Figure 2.1: Asynchronous parallel DE with improved heuristics (`asynPDE_IH`).

figure), while other processes (processes ID 2 and ID 4 in the figure) are in the migration phase. When a process is not able to attend to a migration reception, the message will be stored in the process as a pending migration, avoiding the blocking of the sender process (see process ID 1 in the figure, attending pending migrations).

2.5. Experimental results

This section assesses the impact of the described optimization techniques in the DE algorithm. In order to evaluate the efficiency of the proposed cooperative asynchronous algorithm, different experiments have been carried out. Its behaviour, in terms of convergence and total execution time, was compared with alternative versions of DE. To simplify the understanding of this section, we use the following nomenclature:

- `seqDE`: sequential classic version of DE (Algorithm 2).
- `asynPDE`: proposed asynchronous parallel version of the `seqDE` (Algorithm 3).
- `synPDE`: synchronous parallel version of the `seqDE` (used for comparative purposes).
- `seqDE_IH`, `asynPDE_IH` and `synPDE_IH`: versions of the above algorithms with improved heuristics enabled (local search, tabu list and logarithmic search).

The experiments performed are presented in two subsections. The first one analyzes the performance of the asynchronous parallel cooperative strategy proposed considering an algebraic black-box optimization test-bed. The second one evaluates the overall improvement in a set of computational systems biology problems by combining diversification achieved by the parallel cooperative strategy with the intensification proposed for the local search.

Regarding methodology to carry out the computational experimental runs, both *vertical* and *horizontal* views [87] are analyzed in this section. A vertical approach assesses the performance of a fixed number of evaluations, i.e., a predefined effort; while an horizontal view assesses the performance by measuring the time needed to reach a given target value. Our main goal is to improve the horizontal approach, however, the vertical approach will also benefit from the proposed solution, which is interesting for many real world applications where the total number of evaluations is limited.

In order to enable the reproducibility of these experiments and the comparison with other parallel approaches [216], sufficient information to allow the replication of the different experiments, such as all the configuration parameters used in each test, are provided in next subsections. Also, because of the stochastic properties inherent in these algorithms, several independent runs have been made for each experiment. The number of independent runs and statistical data corresponding to the obtained results are reported as well.

2.5.1. Performance evaluation of the asynchronous parallel strategy

The quality of the solution for the proposed parallelization has to be evaluated, since the proposal implements not only a variant of the sequential DE, the island-model, but also a modification of the classic synchronous implementation of this model. Therefore, it is interesting to determine whether the proposed algorithm challenges the classic implementations in terms of the number of evaluations needed to achieve the required quality solution. In order to obtain a fair comparison between the proposal (`asynPDE`) and the serial classic DE (`seqDE`), the enhancements proposed in Section 2.4, that is, the logarithmic search, the local solver, and the tabu list, were disabled.

The BBOB-2009 data set [87] has been used as a benchmarking testbed due to its popularity and accessibility. This data set is composed of 24 noiseless benchmark functions to be minimized. Although tests have been carried out using the whole BBOB-2009 data set, only five benchmark functions have been selected to illustrate the experimental results in this chapter.

Table 2.1 shows the five selected benchmarks and some of the configuration parameters used in the following experiments. There are many configurable parameters in the classic DE algorithm, such as the mutation scaling factor (F), the crossover

Table 2.1: Subset of BBOB-2009 benchmark functions

Bench.	**Function**	**D**	**NP**	**CR**	**F**	**MSt**	**VTR**
f8	Rosenbrock Function	12	150D	.8	.9	DE/rand/2	149.15
f15	Rastrigin Function	5	150D	.8	.9	DE/rand/2	1000
f17	Schaffers F7 Function	6	150D	.8	.9	DE/rand/2	-16.94
f19	Composite Griewank-Rosenbrock Function F8F2	4	3000D	.9	.9	DE/rand/1	-102.55
f20	Schwefel Function	6	150D	.8	.9	DE/rand/2	-546.5
f22	Gallagher's Gaussian 21-hi Peaks Function	10	150D	.8	.9	DE/rand/2	-1000

constant (CR) or the mutation strategy (MSt), whose selection may have a great impact in the algorithm performance. Other configuration settings in the table are also the dimension (D), the population size (NP) and the value-to-reach (VTR). Since the objective of this chapter is not to evaluate the impact of these parameters, only results for one configuration are reported here. Previous tests have been done to select those parameters that lead to reasonable computation times. Nevertheless, the proposal can be applied to any other configuration parameters. Also, it is worth noting that further performance improvements can be achieved by further fine-tuning settings.

For the selection of the settings in these experiments, in general, the suggestions in [194] have been followed. Regarding F and CR, the settings $F \in [0.5, 1]$ and $CR \in [0.8, 1]$ are recommended. Thus, in these experiments F=0.9 and CR=0.8 have been chosen. Concerning the mutation strategy, among those suggested in [194], namely DE/best/1, DE/best/2, DE/rand/1 and DE/rand/2, the last one has been chosen since some of these benchmarks are multimodal, where this strategy remained most competitive and slightly faster to converge to the value to reach [48]. For the population size, although in [194] a guideline is given where a setting of the DE population size to about ten times the dimensionality of the problem is proposed, this indication is not confirmed in recent studies, such as in [149] where it is shown that a population size lower than the dimensionality of the problem can be optimal in many cases. Since the parallel algorithm will scatter the population matrix between the number of processors, and taking into account that when the population size is small the probability of premature convergence and stagnation may be higher, a $NP = (5 \times D) \times 30 = 150 \times D$ has been chosen for these experiments, where

Table 2.2: Parallel implementation parameters in `asynPDE`.

Parameter	**Value**	**Description**
Island size (λ)	NP/nproc	Size of population matrix for each parallel process.
Migration frequency (μ)	12.5% (every 8 iterations).	Number of iterations to enter in migration stage.
Topology (Tp)	Ring or star.	Communication topology between processors.
Selection policy (SP)	Random (RR) or best (RB).	It selects a set of elements to send in migration stage.
Replacement policy (RP)	Random (RR) or worst (RW).	It replaces a set of elements in subpopulation matrix.
Local configuration (LC)	Homogeneous or Heterogeneous.	It explains how is the value of CR and F for each processor.

the number of processors range between 5 and 60. The only exception was function *f19-Composite Griewank-Rosenbrock Function F8F2*, which is a highly multimodal function that, with the previous settings, frequently fall into an undesired stagnation condition. Therefore, to prevent stagnation in this function, NP has been augmented to 3000 $\times$ D, CR has been set to 0.9, and MSt to DE/rand/1.

In parallel island DE algorithms, new parameters have also to be considered (see Table 2.2), such as migration frequency (μ), island size (λ), communication topology (Tp) between processors, or selection policy (SP) and replacement policy (RP) in the migration step. In SP and RP policy, there are two possible values: random mode (RR) when individuals are selected or replaced randomly, and better/worse mode (BW) when the better individuals are selected to be sent and the worst individuals are replaced.

In addition, the proposed parallel algorithms were tested using different combination of CR and F values in different processes, which enhances diversity as well [228, 229]. Experiments combining these parameters have been thoroughly performed. Local configuration (LC) parameter manages this property: it can be homogeneous (Homo), when all processes have the same attributes, and heterogeneous (Hete) in the opposite case. In heterogeneous case, the values used for the results reported were F={0.9, 0.7} and CR={0.9, 0.7, 0.2}. The combination policy is the following: the algorithm assigns to each process one pair F-CR belonging to the

Table 2.3: asynPDE vs seqDE in function *f-22*

Algorithm	**proc.**	**%hit**	**avg(bestx)**	**#evals**	**time±std(s)**	**speedup**
seqDE	1	4	-999.4315	3001950	10.199 ± 0.037	-
asynPDE	5	20	-999.5550	2937648	1.688 ± 0.086	6.04
	10	44	-999.7322	2841867	0.979 ± 0.097	10.42
	20	60	-999.8317	2744889	0.481 ± 0.055	21.21
	40	78	-999.9434	2576759	0.246 ± 0.039	41.36
	60	92	-999.9987	2537147	0.187 ± 0.023	54.34

sorted list of all possible combinations of the above sets of F and CR. When all the pairs have been distributed, it restarts the previous sorted list and the algorithm continues the distribution.

In the experiments performed with BBOB-2009 benchmarks, it is important to note that the global population has been distributed among the cores. This assures that the improvement in the diversity comes from the asynchronous sparse individual exchanges in the parallel version, allowing to analyze the impact of this technique on the algorithm computational efficiency. So, the island size for these experiments is $NP/PROC$.

Experiments, consisting of 50 independent runs each, were carried out on Intel Sandy Bridge nodes of CESGA SVG Linux cluster [40]. Table 2.3 shows, for the *f-22 Gallagher's Gaussian 21-hi Peaks Function* of the BBOB-2009 data set, the percentage of executions that achieve the optimal value (% hit), the average value of the achieved tolerances (avg bestx), the mean number of evaluations (#evals) and the mean execution time. The values of the configuration settings in this case were: the topology was a ring, the SP/RP policies consisted in RR, μ was equal to 12.5%, and it had a Heterogeneous strategy for their LC. Regarding the stopping criteria, it was a combination of two conditions: maximum number of evaluations equal to 3000000 or achieve a VTR near to -1000 with an absolute tolerance TOL=1e-3. Results for the serial DE (seqDE) and for the proposed parallel island-based algorithm (asynPDE) with different number of cores are shown. First, it can be observed that when the number of cores grows (P-5 to P-60 in Table 2.3), the number of the executions that achieve the quality solution increases. It can also be observed that the number of evaluations needed to achieve the optimal solution decreases when the number of processors grows. In short, these results show the effectiveness of the parallel algorithm proposed in terms of quality of the solution, where less evaluations are needed to come to the required tolerance when more processors take part in the search. Table 2.3 also shows speedups for the proposed algorithm `asynPDE` versus the `seqDE` with different number of processors. At first glance, it can be observed that the speedups obtained are superlinear. That is due to the diversity introduced by the migration phase in the parallel execution, that actually improves the effectiveness of the DE algorithm.

Two different stopping criteria for the algorithms were considered in these experiments: solution quality, for an horizontal view, and maximum effort, for a vertical approach. Thus, two different speedup definitions were used to compare the sequential and the parallel algorithm: a speedup with solution quality stop, and a speedup with predefined effort. The first one shows how fast the parallel algorithm reaches DE solution versus the sequential algorithm. This is the case of the results shown in Table 2.3. This speedup is due both to the distribution of computations among the processors, and to the effectiveness of the parallel algorithm, which requires fewer iterations as the number of processors grows.

However, in most of the cases, the sequential algorithm and the parallel one running on few processors do not reach the quality solution before the maximum allowed effort. In those cases, the best way to fairly compare sequential and parallel executions is to stop all of them at a predefined effort, that is, for a predefined number of evaluations. Since the number of iterations performed are, in this case, the same, these speedup results help to analyze how much faster are the iterations of the parallel algorithm versus the classic iterations.

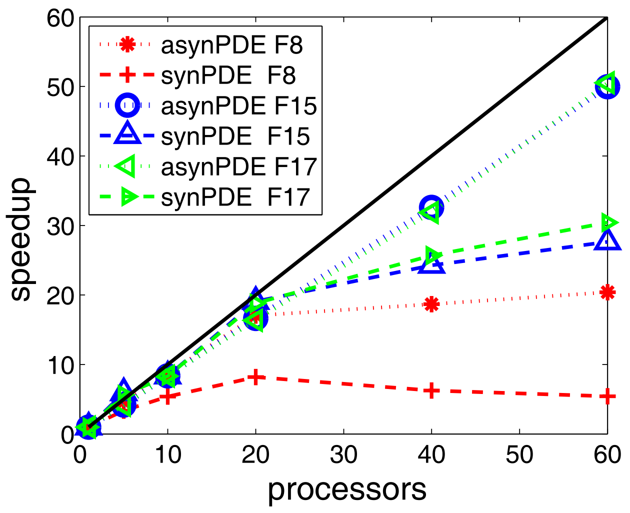
Table 2.4 shows the speedup with quality solution for those configurations that reach the required tolerance in a reasonable amount of time and eventually obtain the best speedups. Replace and selection policies were RR, and the value-to-reach (VTR) in each experiment, generated by the BBOB benchmark software, is shown in Table 2.1 with an absolute tolerance TOL=1e-3. Table 2.4 shows the mean and standard deviation of `seqDE` runtimes, the LC and Tp parameters that obtained the best results for each parallel algorithm (`asynPDE` or `synPDE`), and speedup results calculated versus the `seqDE` implementation. It is noticeable that, in most cases, the heterogeneous strategies are superior to homogeneous ones, since they benefit from search diversification.

Table 2.4: Quality value test in BBOB benchmarks.

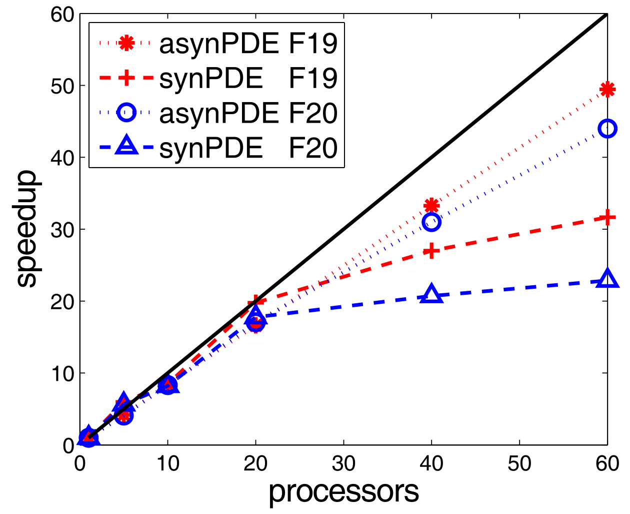
Func.	**seqDE** time±std (s)	**Alg.**	**LC**	**Tp**	**speedup** Number of processors **5**	**10**	**20**	**40**	**60**
f8	13.55±1.17	`asynPDE`	Hete	Ring	5.2	10.8	21.1	34.0	43.8
		`synPDE`	Hete	Ring	5.0	9.5	16.9	15.4	13.2
f15	20.41±0.73	`asynPDE`	Hete	Ring	4.3	8.7	43.4	171.9	221.9
		`synPDE`	Homo	Ring	9.8	22.3	69.0	123.1	136.4
f17	7.26±0,35	`asynPDE`	Hete	Star	6.7	12.6	26.1	70.7	120.8
		`synPDE`	Hete	Star	7.6	15.9	29.2	56.4	68.3
f19	26.93±10.39	`asynPDE`	Homo	Ring	7.3	11.2	40.9	124.8	155.4
		`synPDE`	Hete	Ring	9.0	18.6	45.3	95.4	128.0
f20	18.12±0,88	`asynPDE`	Hete	Ring	21.3	31.9	113.3	318.6	447.2
		`synPDE`	Hete	Ring	21.8	50.1	131.7	235.2	252.0

Figure 2.2 shows the speedup from a vertical view, that is, for a predefined effort. These experiments were configured with the following parallel implementation parameters: LC consisted in a heterogeneous strategy, Tp=ring, SP/RP=RR and μ was equal to 12,5%. Results for both the proposed asynchronous algorithm (`asynPDE`) and for a synchronous version of the classic DE (`synPDE`) are shown for comparison purposes. It can be seen that, for a small number of processors, the speedup achieved when using the stopping criterion of predefined effort is similar in synchronous and asynchronous strategies, indicating that the evaluation time is similar in both. For a small number of processors, the synchronous version obtains higher speedups from the horizontal view, that is, using the solution quality as stopping criterion (data shown in Table 2.4). This is because fewer evaluations are needed to reach the solution in the synchronous version. The reason is that, when the processes are synchronized in the migration phase, the new evaluations are performed after exchanging solutions between processors. In the asynchronous version, however, the new evaluations after the migration point can start before the complete exchange of solutions, since no synchronization is forced between processes.

As the number of processors increases, it can be observed that the scalability of asynchronous proposal improves versus the synchronous one. This is because the synchronization slows down the processes, since it implies more processes' stalls while waiting for data. Thus, the asynchronous version, in spite of requiring further evaluations, reaches the solution in a shorter time.



(a) $f8$, $f15$ and $f17$ functions



(b) $f19$ and $f20$ functions

Figure 2.2: Predefined effort test in BBOB benchmarks. Stopping criteria: maximum number of evaluations=1000200.

In summary, these results show that the proposed asynchronous island-based algorithm achieves good speedup, and it scales better than the synchronous algorithm for a large number of processors.

2.5.2. Results for parameter estimation problems in systems biology

In order to evaluate the global algorithm proposed in this chapter (`asynPDE_IH`), that combines the diversification introduced by the parallelization evaluated previously and the intensification of the local search three challenging parameter estimation problems from the domain of computational system biology were considered. These problems are known to be particularly hard due to their ill-conditioning and non-convexity [141, 218]:

- *Circadian* model: parameter estimation in a dynamic model of the circadian clock in the plant *Arabidopsis thaliana*, as presented in [127]. The model consists of 7 ordinary differential equations (ODEs) with 27 parameters (13 of them were estimated) with data sets from 2 experiments.
- *NF-κB* model: this problem is based on the model in [126] and consists of 15 ordinary differential equations with 29 parameters and data sets from 2 experiments.
- *3-step pathway model*: problem considering a 3-step generic and highly non-linear pathway with 8 differential equations and 36 parameters, and data sets from 16 experiments, as presented in [141].

The aim of these experiments is to demonstrate the potential of the proposed techniques for improving the convergence and execution time of very hard problems. Since the goal of including the enhancements proposed in Section 2.4 in the DE algorithm is to improve the effectiveness of each local evaluation, it is desirable to enhance the exploration in the solution space by means of increasing the diversification in the parallel threads. Therefore, for these experiments, the heterogenous LC was used in all the cases.

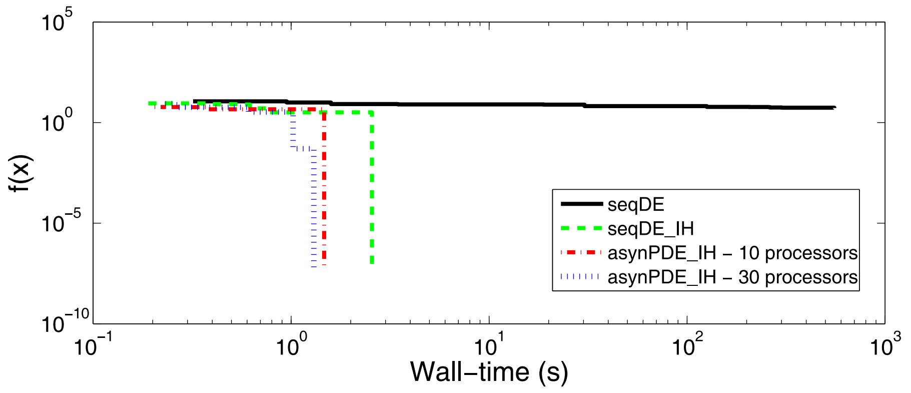
The multicore cluster Pluton was used to carry out these experiments. It consists of 16 nodes powered by two octa-core Intel Xeon E5-2660 CPUs with 64 GB of RAM. The cluster nodes are connected through an InfiniBand FDR network. OpenMPI library version 1.6.2 has been used to compile the parallel implementations, and the experiments were carried out using from 1 to 30 processors. In these benchmarks, the execution of only one test of `seqDE` could take hours or even days to complete. For each experiment 30 independent runs have been performed.

Figure 2.3 shows the best convergence curve for these three parameter estimation problems. The settings for these experiments were the followings: NP=10$\times$D, MSt=DE/rand/2, LC was heterogeneous, the topology was a ring, SP=RB, RP=RW, μ=33%, λ=NP/NPROC, the local solver condition was L=6 and the stopping criteria was achieved to a VTR (value to reach)=1e-5 for (a) and (b), and VTR=1e-2 for (c). Results show that, on the one hand, the logarithmic search, the local search, and the tabu list, on `seqDE_IH` achieve by themselves a drastic reduction in the execution time required for convergence, due to a reduction in the number of evaluations needed; on the other hand, the parallelization on `asynPDE_IH` also improves quality of mean solution since more evaluations are performed in the same amount of time.

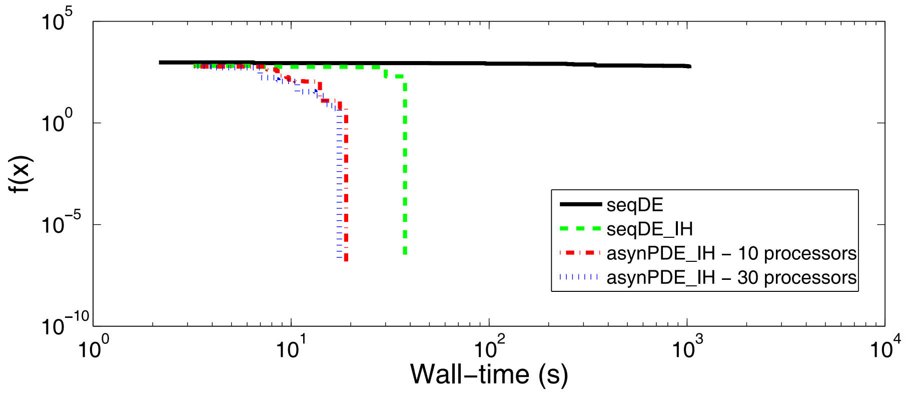
Figure 2.4 illustrates how the proposed `asynPDE_IH` reduces the big variability of execution time in the sequential version of the algorithm. It demonstrates that when the number of processors increase in the asynchronous method, the outliers of the execution time decrease. This is an important feature in the performance of this solver, because it reduces the average execution time for each benchmark.

Figure 2.5 shows the speedup for a quality value test in these three parameter estimation problems. These speedups were calculated comparing execution times of `synPDE_IH` and `asynPDE_IH` with `seqDE_IH`. Note that the `seqDE` execution time cannot be used for comparison, because of its unreasonable amount of time to converge (more than 48 hours for each 3-step and NF-κB experiment, when they do not get stuck in local optima). These figures show that the proposed `asynPDE_IH` significantly reduces execution time required, and scales better than the `synPDE_IH` algorithm for a large number of processors.

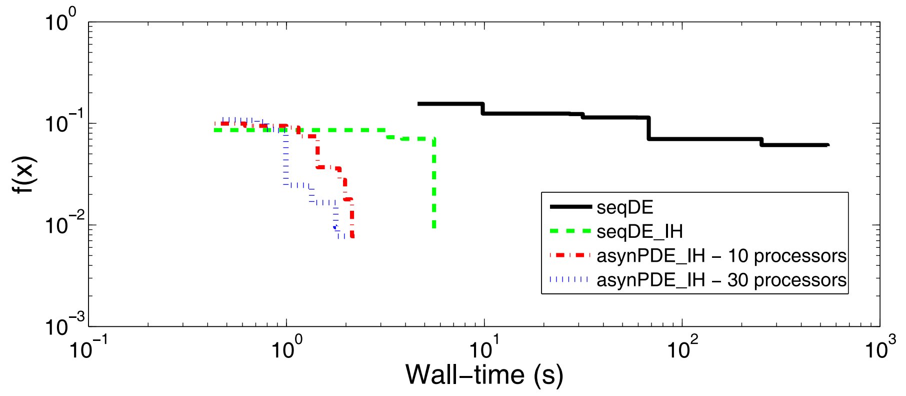
The speedup results in these figures may appear to be modest, since they have been calculated related to the `seqDE_IH` execution time instead of to the `seqDE` execution time. Table 2.5 shows results for both the execution time and the number of evaluations performed in these experiments. Note that the number of evaluations reported in this table includes those performed by the local solver. Although each external iteration of the algorithm involves more evaluations when the local solver



(a) Circadian problem

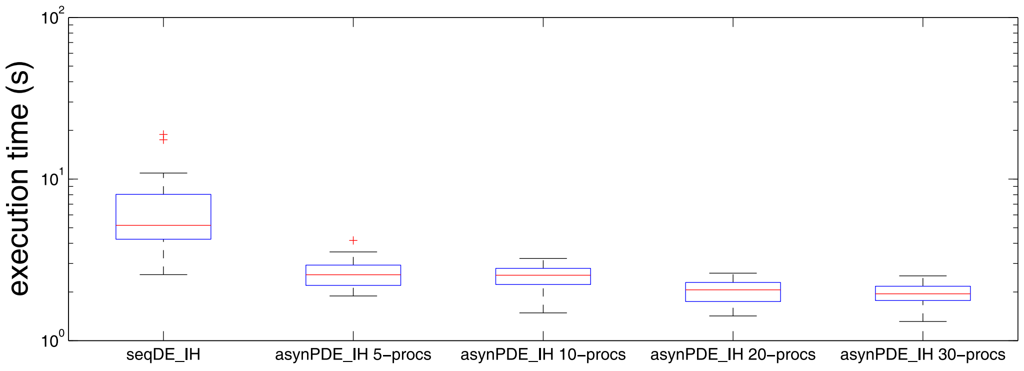


(b) 3-steps pathway problem

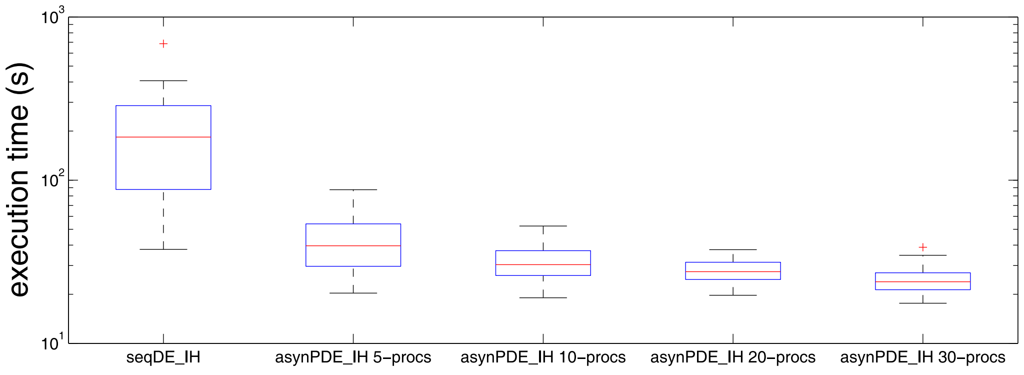


(c) NF-κB problem

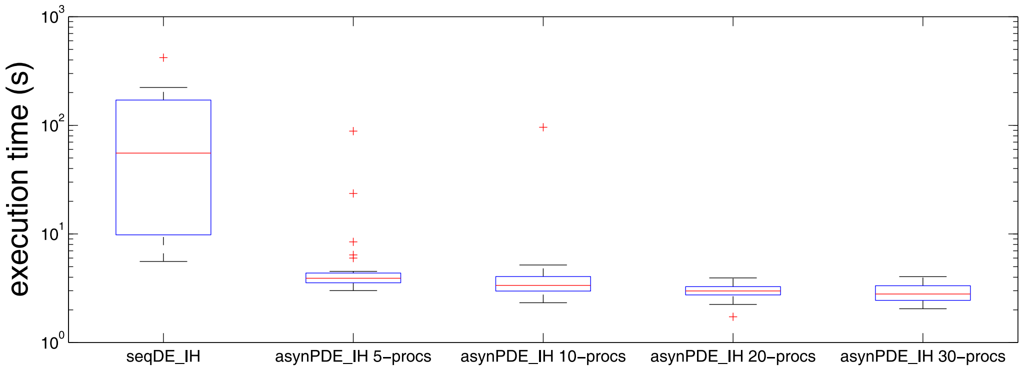
Figure 2.3: Best convergence curves for parameter estimation problems.



(a) Circadian problem

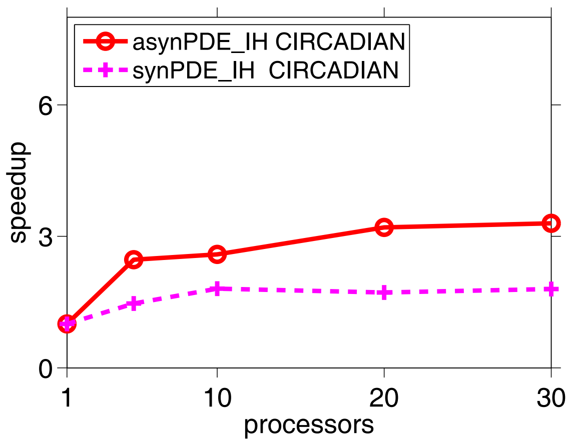


(b) 3-steps pathway problem

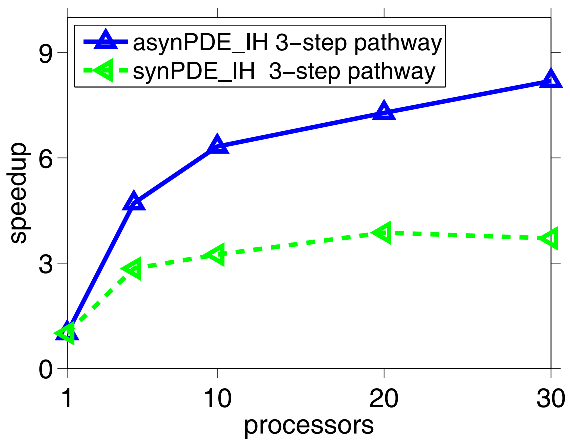


(c) NF-κB problem

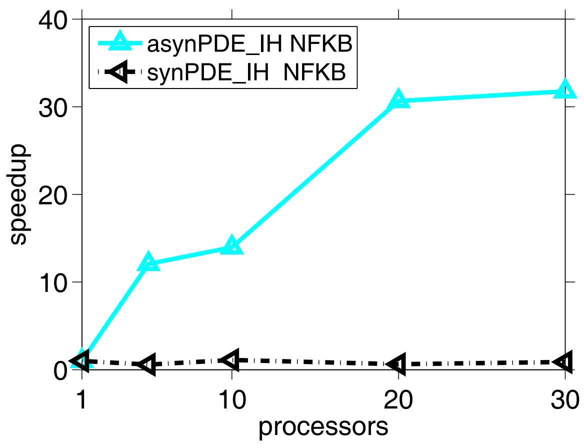
Figure 2.4: Boxplot of the execution time for parameter estimation problems with `asynPDE_IH`.



(a) Circadian problem



(b) 3-steps pathway problem



(c) NF-κB problem

Figure 2.5: Speedups calculated with respect to the execution time of `seqDE_IH`.

Table 2.5: Execution time table of the experiments in Figure 2.5.

prob.	algorithm	proc	#evals	#evals/proc	#iter	time±std(s)
circadian	`seqDE`	1	11521835	11521835	88627.7	26856.12±3424.14
	`seqDE_IH`	1	3534.37	3534.37	11.20	6.46±3.81
	`asynPDE_IH`	10	10382.37	1038.23	7.10	2.50±0.27
	`asynPDE_IH`	20	17398.73	869.93	6.70	2.02±0.32
	`asynPDE_IH`	30	23935.47	797.85	6.58	1.96±0.43
3-step pathway	`seqDE`	1	-	-	-	-
	`seqDE_IH`	1	30580.97	30580.97	32.60	202.40±138.53
	`asynPDE_IH`	10	43802.94	4380.29	11.03	32.01±8.77
	`asynPDE_IH`	20	73970.33	3698.57	9.28	27.79±4.74
	`asynPDE_IH`	30	95001.10	3166.70	8.20	24.67±5.06
NF-κB	`seqDE`	1	-	-	-	-
	`seqDE_IH`	1	11253.90	11253.90	34.00	91.37±98.68
	`asynPDE_IH`	10	4370.10	437.01	8.30	6.55±16.90
	`asynPDE_IH`	20	4688.27	234.41	7.31	2.98±0.52
	`asynPDE_IH`	30	5839.67	194.66	6.76	2.88±0.56

is enabled, the overall result shows that the enhanced algorithm requires less total number of evaluations to reach the optimum value.

In these problems, the improvement introduced by the local solver achieves convergence with very few external iterations of the algorithm, that is, with very few migration phases between islands. Therefore, adding more processes does not significantly improve the execution time. In the NF-κB problem, the convergence improvement achieved by the local solver is not that noticeable, thus, the parallelization continues introducing diversity when the number of processes increases so performance enhancement is achieved. In all the cases, when the number of external iterations are drastically diminished, there is no room for further improvement despite the increased number of processors.

It is important to note that the local solver introduces a great overhead on the execution of each evaluation. Moreover, it is responsible for the lack of synchronization between processes at the migration step, since it leads processes into a more computationally unbalanced scenario, thus, giving advantage to the proposed asynchronous solution. When more evaluations are required, the synchronization

Table 2.6: Wilcoxon signed ranks test with a significance level α=0.05.

prob.	**algorithms**	R^+	R^-	p**-value**
circadian	`asynPDE_IH` 10-procs vs `seqDE_IH`	465	0	1.73E-06
	`asynPDE_IH` 30-procs vs `asynPDE_IH` 10-procs	424	41	8.19E-05
	`asynPDE_IH` 10-procs vs `synPDE_IH` 10-procs	446	19	1.13E-05
	`asynPDE_IH` 30-procs vs `synPDE_IH` 30-procs	464	1	1.92E-06
3-step pathway	`asynPDE_IH` 10-procs vs `seqDE_IH`	465	0	1.73E-06
	`asynPDE_IH` 30-procs vs `asynPDE_IH` 10-procs	399	66	6.16E-04
	`asynPDE_IH` 10-procs vs `synPDE_IH` 10-procs	443	22	1.49E-05
	`asynPDE_IH` 30-procs vs `synPDE_IH` 30-procs	465	0	1.73E-06
NF-κB	`asynPDE_IH` 10-procs vs `seqDE_IH`	465	0	1.73E-06
	`asynPDE_IH` 30-procs vs `asynPDE_IH` 10-procs	370	95	4.70E-03
	`asynPDE_IH` 10-procs vs `synPDE_IH_`IH 10-procs	441	24	1.80E-05
	`asynPDE_IH` 30-procs vs `synPDE_IH` 30-procs	465	0	1.73E-06

between processes will decrease, thus, better performance will be achieved by the asynchronous proposal. This is the case of the NF-κB problem, that requires more evaluations to converge than the other two benchmarks.

To prove the statistical significance of the results, the Wilcoxon Rank-sum test has been applied for a confidence level of 0.95. Table 2.6 shows the results of the test, using the runtimes obtained in the experiments. The parameters in the table R^+ and R^- are the sum of the positive/negative ranks. As can be seen, the p-value is always smaller than the significance, thus, `asynPDE_IH` outperforms both the `seqDE_IH` and the `synPDE_IH`. Note also that, when comparing the asynchronous proposal with the synchronous one, the values of p-value are higher when using few processors and they decrease when the number of processors grows. This demonstrates the scalability of the proposal over the synchronous version.

Compared to the serial DE (`seqDE`), the proposed enhancements improve the execution time in several orders of magnitude. For the circadian problem, the execution of the `seqDE` lasts more than 7 hours while the proposed `seqDE_IH` requires only a few seconds, which demonstrates the potential of the proposed heuristics in the solution on these problems.

2.6. Concluding remarks

In this chapter, we presented an improved Differential Evolution algorithm designed to solve complex problems in computational systems biology. The key idea is to achieve a proper balance of the exploration abilities of DE and the exploitation abilities of efficient local search. The method improves global search through an asynchronous parallel implementation based on a cooperative island-model. The improved local search is implemented by means of several heuristics (efficient local solver, tabu list, logarithmic search) which exploit the structure of parameter estimation problems in systems biology, the main application area considered here.

It should be noted that, although the method presented here is based on a parsimonious hybrid (global-local) design, the three heuristic enhancements introduced in the local search are fundamental to successfully exploit the special characteristics of these systems biology problems, which are typically very ill-conditioned and highly multimodal, as reviewed in [218]. The results obtained show that this improved local search mechanism, combined with the parallel cooperation scheme, allow an adequate balance between exploration and exploitation for the class of problems considered.

The experimental results show that (i) convergence time can be reduced by several orders of magnitude when the local search heuristics are included in the DE algorithm, and (ii) the asynchronous parallel strategy proposed attains a further reduction in the convergence time through collaboration of the parallel processes, demonstrating also a competitive speedup against the synchronous approaches.

As an example of the practical significance of this proposal, one of the systems biology benchmarks considered, the 3-steps pathway problem, typically requires more than 3 days of computation time using the classic version of DE executed in one of the cores of our test machine, but it can be solved in less than one minute with the novel asynchronous parallel method presented.

Although the improved DE was designed and tested with focus on the field of parameter estimation problems in computational systems biology, it can also be directly applied to solve arbitrary global optimization problems.

The results of this chapter have been published in:

- D. R. Penas, J. R. Banga, P. González, and R. Doallo. A parallel differential evolution algorithm for parameter estimation in dynamic models of biological systems. *Advances in Intelligent Systems and Computing*, 294:173–181, 2014. Proceedings of the 8th International Conference on Practical Applications of Computational Biology & Bioinformatics (PACBB 2014). [164]

- D. R. Penas, J. R. Banga, P. González, and R. Doallo. Enhanced parallel differential evolution algorithm for problems in computational systems biology. *Applied Soft Computing*, 33:86–99, 2015. [165]

The source code of the asynPDE proposed here (see Appendix A) is made publicly available at:

- `https://bitbucket.org/DavidPenas/asynpde`

Chapter 3

Self-adaptive Cooperative enhanced Scatter Search

Scatter Search (SS) is a promising population-based metaheuristic for solving combinatorial and nonlinear optimization problems. This evolutive method uses strategies to combine solution vectors stored in a small population to obtain good results without spending many resources. Different implementations have been shown [22, 50, 61, 71, 74, 94, 104, 181] where SS can outperform other state of the art stochastic global optimization methods. However, as already commented upon for the DE algorithm, current realistic applications are very complex, and SS still requires a very large computation time to obtain a good quality solutions.

This chapter aims to improve this method by proposing a new parallel cooperative scheme that incorporates:

- an improved cooperative scheme, including an information exchange mechanism driven by the quality of the solutions.
- an asynchronous communication protocol to handle inter-process information exchange.
- the combination of a coarse-grained distributed-memory parallelization paradigm and an underlying fine-grained parallelization of the individual tasks with a shared-memory model, in order to improve the scalability.

- self-adaptive procedures to dynamically tune the settings of the parallel searches during the execution progress.

The performance and scalability of this novel method is illustrated considering a set of very challenging parameter estimation problems in large-scale dynamic models of biological systems. These problems consider kinetic models of the bacterium *E. coli*, baker's yeast *S. cerevisiae*, the vinegar fly *D. melanogaster*, chinese hamster ovary cells and a generic signal transduction network. The results consistently show that cooperative strategies of SS are robust and efficient methods, allowing for a very significant reduction of computation times with respect to previous methods (from days to minutes, in several cases) even when only a small number of processors is used. Therefore, we believe that this new method can play a key role in the development of large-scale dynamic models in systems biology.

The organization of this chapter is as follows. Section 3.1 describes a basic implementation of Scatter Search, taken as reference for the rest of this chapter. Then, Section 3.2 covers the related work, and Section 3.3 describes the self-adaptive cooperative enhanced Scatter Search (saCeSS) proposed in this chapter. Section 3.4 shows the performance and scalability results of the proposed method when it is applied to a set of challenging problems in computational systems biology. Finally, Section 3.5 summarizes the main conclusions of this chapter.

3.1. Scatter Search algorithm

Scatter Search (SS) [81] is a population based metaheuristic that constructs new solutions based on systematic combinations of the members of a reference set (called *RefSet* in this context). The *RefSet* is the analogous concept to the *population* in genetic or evolutionary algorithms but its size is considerably smaller than in those methods. As a consequence the degree of randomness in SS is lower than in other population based metaheuristics and the generation of new solutions is based on the combination of the *RefSet* members. Another difference between SS and other classic population based methods is the use of a so-called *improvement method*, which usually consists of local searches from selected solutions to accelerate the convergence to the optimum in certain problems, turning the algorithm into a more

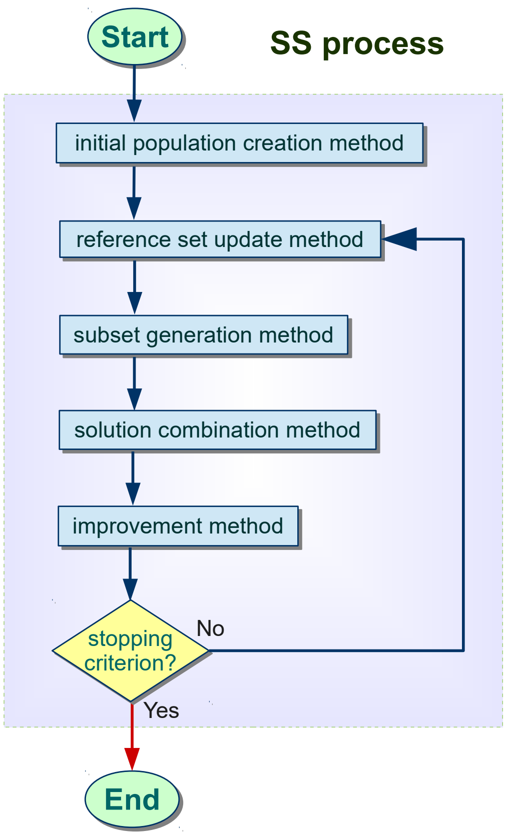

Start
SS process
initial population creation method
reference set update method
subset generation method
solution combination method
improvement method
stopping criterion?
No
Yes
End

Figure 3.1: Schematic representation of a basic Scatter Search algorithm.

effective combination of global and local search. This *improvement method* can of course be ignored in those problems where local searches are very time-consuming and/or inefficient.

Figure 3.1 shows a schematic representation of a basic SS algorithm where the steps of the popular *five-step template* [80] are highlighted. Classic SS implementations update the *RefSet* by replacing the worst elements with new ones which outperform their quality. In continuous optimization, as is the case of the problems considered in the present study, this can lead to premature stagnation and lack of diversity among the *RefSet* members. The SS version used in this chapter as a starting point is based on a recent implementation [60, 61], named enhanced Scatter Search (eSS), in which the population update is carried out in a different way in order to avoid stagnation problems and increase the diversity of the search without losing efficiency.

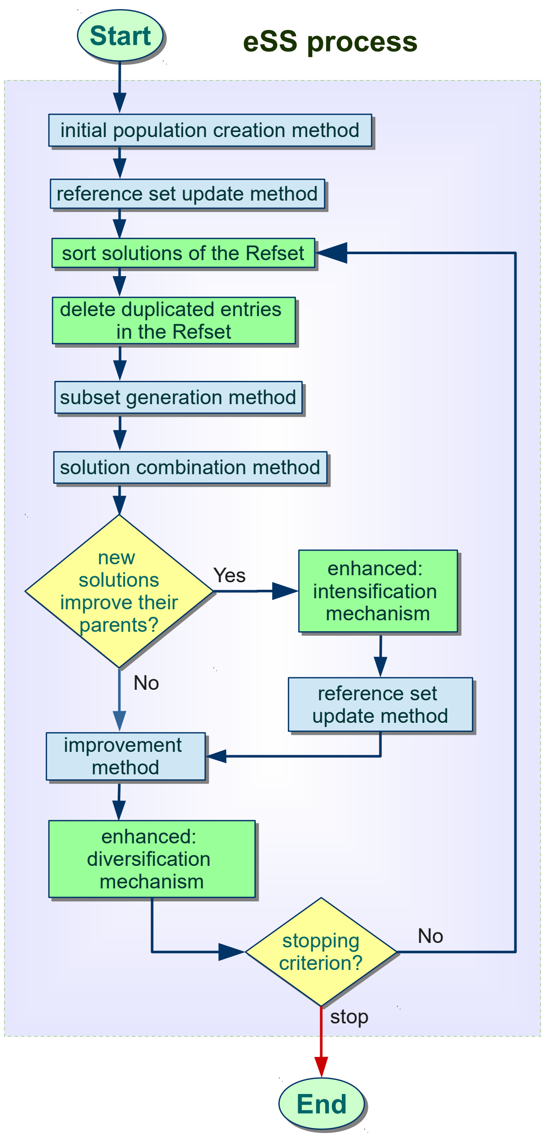

Start
eSS process
initial population creation method
reference set update method
sort solutions of the Refset
delete duplicated entries in the Refset
subset generation method
solution combination method
new solutions improve their parents?
Yes
enhanced: intensification mechanism
reference set update method
No
improvement method
enhanced: diversification mechanism
stopping criterion?
No
stop
End

Figure 3.2: Schematic representation of the enhanced Scatter Search algorithm.

Figure 3.2 schematically illustrates the eSS algorithm. Basic pseudocodes of the eSS algorithm are shown in Algorithm 5 (main routine), Algorithm 6 (go-beyond strategy) and Algorithm 7 (local search). The method begins by creating and evaluating an initial set of *ndiverse* random solutions within the search space (line 4, Algorithm 5). Then, the *RefSet* is generated using the best solutions and random solutions from the initial set (line 6). When all data is initialized and the first *RefSet* is created, the eSS repeats the main loop until the stopping criterion is fulfilled.

These main steps of the algorithm are briefly described in the following lines:

1. *RefSet* order and duplicity check: The members of the *RefSet* are sorted by quality. After that, if two (or more) *RefSet* members are too close to one another, one (or more) will automatically be replaced by random solutions (lines 8-12). These comparisons are performed pair to pair for all members of the *Refset*, considering normalized solutions: every solution vector is normalized in the interval [0, 1] based on the upper and lower bounds. Thus, two solutions are "too close" to each other if the maximum difference of its components is higher than a given threshold, with a default value of 1e-3. This mechanism contributes to increase the diversity in the *RefSet* thus preventing the search from stagnation.

2. Solution combination: This step consists in pair-wise combinations of the *RefSet* members (lines 13-23). The new solutions resulting from the combinations are generated in hyper-rectangles defined by the relative position and distance of the *RefSet* members being combined. This is accomplished by doing linear combinations in every dimension of the solutions, weighted by a random factor and bounded by the relative distance of the combined solutions. More details about this type of combination can be found in [60].

3. *RefSet* update: The solutions generated by combination can replace the *RefSet* members if they outperform their quality (line 28). In order to preserve the *RefSet* diversity and avoid premature stagnation, a $(1+\lambda)$ evolution strategy [27] is implemented in this step. This means that a new solution can only replace that *RefSet* member that defined the hyper-rectangle where the new solution was created. In other words, a solution can only replace its "parent". Moreover, among all the solutions generated in the same hyper-rectangle, only the best of them will replace the "parent". This mechanism avoids clusters of similar solutions in early stages of the search which could produce premature stagnation.

4. Enhanced mechanisms: eSS includes two additional procedures to make the search more efficient. One is the so-called "go-beyond" strategy (labeled as

intensification mechanism in the figure) and consists in exploiting promising search directions. If a new solution outperforms its "parent", a new hyper-rectangle following the direction of both solutions and beyond the line linking them is created. A new solution is created in this new hyper-rectangle, and the process is repeated varying the hyper-rectangle size as long as there is improvement (Algorithm 6). The second mechanism consists in a stagnation checking (labeled as *diversification mechanism* in the figure). If a *RefSet* solution has not been updated during a predefined number of iterations, we consider that it is a local solution and replace it with a new random solution in the *RefSet*. This is carried out by using a counter (n_{stuck}) for each *RefSet* member (lines 29-34, Algorithm 5).

5. Improvement method. This is basically a local search procedure that is implemented in the following form (see Algorithm 7): when the local search is activated, we distinguish between the first local search (which is carried out from the best found solution after *local.n1* function evaluations), and the rest. Once the first local search has been performed, the next ones take place after *local.n2* function evaluations from the previous local search. In this case, the initial point is chosen from the new solutions created by combination in the previous step, balancing between their quality and diversity. The diversity is computed measuring the distance between each solution and all the previous local solutions found. The parameter *balance* gives more weight to the quality or to the diversity when choosing a candidate as the initial point for the local search. Once a new local solution is found, it is added to a list. There is an exception when the *best_sol* parameter is activated. In this case, the local search will only be applied over the best found solution as long as it has been updated in the incumbent iteration. Based on our previous experience, this strategy is only useful in certain pathological problems, and should not be activated by default.

For further details on the eSS implementation, the reader is referred to [60, 61].

Algorithm 5: Basic pseudocode of eSS.

```
1  Set parameters: $dim\_refset$, $best\_sol$, $local.n1$, $local.n2$, $balance$, $ndiverse$;
2  Initialize $n_{stuck}$, $neval$;
3  $local\_solutions = \emptyset$;
4  Create set of random $ndiverse$ solutions and evaluate them;
5  $neval = neval + ndiverse$;
6  Generate the initial $RefSet$ with $dim\_refset$ solutions with the best solutions and
     random elements from $ndiverse$;
7  repeat
8      Sort $RefSet$ by quality $RefSet = \{x^1, x^2, \ldots, x^{dim\_refset}\}$ so that $f(x^i) \leq f(x^j)$
         where $i, j \in [1, 2, \ldots, dim\_refset]$ and $i < j$;
9      if $max\Big(abs\left(\frac{x^i - x^j}{x^j}\right)\Big) \leq \epsilon$ with $i < j$ then
10         Replace $x^j$ in the $RefSet$ by a random solution and evaluate it;
11         $neval = neval + 1$;
12     end
13     $\mathbf{y} = \emptyset$;
14     for $i = 1$ to $dim\_refset$ do
15         for $j = 1$ to $dim\_refset$ do
16             if $i \neq j$ then
17                 Combine $x^i$ with $x^j$ to generate a new solution, $y^{i,j}$;
18                 $\mathbf{y} = \mathbf{y} \cup \{y^{i,j}\}$;
19                 Evaluate $y^{i,j}$;
20             end
21         end
22         $neval = neval + dim\_refset - 1$;
23     end
24     Call to go-beyond strategy
25     if local search is activated then
26         Apply local search routine (see Algorithm 7);
27     end
28     Replace labeled $RefSet$ members by their corresponding $y^{i,*}$ and reset $n_{stuck}(i)$;
29     $n_{stuck}(j) = n_{stuck}(j) + 1$ where $j$ is the index of a not labeled $RefSet$ member;
30     for $i = 1$ to $dim\_refset$ do
31         if $n_{stuck}(i) > nchange$ then
32             Replace $x^i \in RefSet$ by a random solution and set $n_{stuck}(i) = 0$;
33         end
34     end
35 until stopping criterion is met;
```

Algorithm 6: Pseudocode of go-beyond strategy.

```
for $i = 1$ to $dim\_refset$ do
    $y^{i,*}$ = best solution in $y^{i,j} \forall j \in [1, 2, \ldots dim\_refset]$;    $j \neq i$;
    if $f(y^{i,*}) < f(x^i)$ then
        Label $x^i$;
        $x_{\text{temp}} = x^i$;
        $improvement = 1$;
        $\Lambda = 1$;
        while $f(y^{i,*}) < f(x_{\text{temp}})$ do
            Create a new solution, $x_{\text{new}}$, in the hyper-rectangle defined by:
              $\left[ y^{i,*} - \frac{(x_{\text{temp}} - y^{i,*})}{\Lambda}, y^{i,*} \right]$;
            $x_{\text{temp}} = y^{i,*}$;
            $y^{i,*} = x_{\text{new}}$;
            $neval = neval + 1$;
            $\mathbf{y} = \mathbf{y} \cup \{y^{i,*}\}$;
            $improvement = improvement + 1$;
            if $improvement = 2$ then
                $\Lambda = \Lambda/2$;
                $improvement = 0$;
            end
        end
        if $f(y^{i,*}) < fbest$ then
            $xbest = y^{i,*}$;
            $fbest = f(y^{i,*})$;
        end
    end
end
```

Algorithm 7: Pseudocode of the local search procedure.

```
1  if $best\_sol$ is activated then
2      if $xbest$ was updated since last iteration then
3          Apply local search over $xbest$;
4      end
5  else
6      if $local\_solutions = \emptyset$ then
7          if $neval \geq local.n1$ then
8              Apply local search over $xbest$;
9          end
10     else
11         if $neval \geq local.n2$ then
12             Sort $\mathbf{y}$ by quality, creating $\mathbf{y_q} = \{y_q^1, y_q^2, \ldots y_q^m\}$ where $f(y_q^i) \leq f(y_q^j)$ if
               $i < j$;
13             Compute the minimum distance between each element $i \in [1, 2, \ldots, m]$
               in $\mathbf{y}$ and all the local optima found so far.
               $d_{min}(i) = \min ||\mathbf{y^i} - local\_solutions||^2$;
14             Sort $\mathbf{y}$ by diversity, creating $\mathbf{y_d} = \{y_d^1, y_d^2, \ldots y_d^m\}$ where
               $d_{min}(i) \geq d_{min}(j)$ if $i < j$;
15             for each solution $y^k \in \mathbf{y}$ do
16                 $score(y^k) = (1 - balance) \cdot i + balance \cdot j$
17                 where $i$ is the index of $y^k$ in $\mathbf{y_q}$ and $j$ is the index of $y^k$ in $\mathbf{y_d}$;
18             end
19             Apply local search over $y^l : score(y^l) = \min score(\mathbf{y})$
20         end
21     end
22 end
23 Produce a local solution $z^*$;
24 if $z^* \notin local\_solutions$ then
25     $local\_solutions = local\_solutions \cup \{z^*\}$;
26 end
27 if $f(z^*) < fbest$ then
28     $xbest = z^*$;
29     $fbest = f(z^*)$;
30 end
31 $neval = 0$
```

3.2. Related work

Several researches have added advanced elements to the basic SS method in order to improve its performance in selected problems. In [181] a hybrid stochastic-deterministic SS method for model calibration in nonlinear dynamic models of biological systems was presented, obtaining a robust and efficient global optimization approach. Furthermore, an extension of the previous work was proposed in [61], called enhanced Scatter Search (eSS), that implements new strategies to intensify the search and to prevent stuck in local optima regions. These enhanced versions of SS have been successfully used in several works, such as in [74, 104], to propose new strategies for parameter estimation; or [50], where multi-objective optimization is used with the aim to study the activation mechanism in metabolic pathways; and also in [22, 94], for reverse engineering problems. In [71] a comparative study of model calibration methods was presented, that demonstrates encouraging results for previous implementations of the SS method.

However, not much research has been done regarding the parallelization of this metaheuristic. In [76] three parallel strategies were explored: a low-level synchronous parallel SS model using parallel search instead of local search, a replicated combination SS model that distributes multiple subsets on the processors, and a natural replication of parallel SS. All these methods were implemented using shared-memory techniques, and, thus, present limitations on scalability. In [32] a parallel algorithm based on SS and path re-linking methods was presented. In this proposal, the master process creates the starting solutions set while calculations of path re-linking are executed by the slave processes on local data. The slaves send the new solutions to the master that creates a new set of starting solutions. Another parallel SS algorithm is presented in [128], based on replacing the combination method by parallel execution of two greedy methods on every processor.

In [219], a cooperative parallel strategy for the eSS method has been presented. The proposed Cooperative enhanced Scatter Search (CeSS) follows a master-slave approach, where each of the slave processors runs a sequential eSS algorithm, while they exchange their reference set of solutions through a master processor at certain fixed instants. The results presented in [219] show how cooperation of individual parallel search slaves modifies the systemic properties of the individual algorithms, improving its performance and outperforming other competitive methods. However, the parallel strategy followed presents serious issues. First, exchanging information at fixed instants, in a blocking fashion, forces a synchronization that delays the progress in many slaves. Processors are idle during a significant amount of time, while they are waiting for each other during the migration steps. The penalty may be significant because the improvements introduced by the eSS in diversification, like the local solver or the use of a tabu list, lead processes into a more computationally unbalanced scenario. This synchronization is one of the causes of its poor scalability when the number of processors grows.

In a preliminary work [166], we have explored the parallelization of SS by means of an asynchronous cooperative parallel strategy (aCeSS). The proposed aCeSS method improves the CeSS algorithm in [219] by means of a cooperative scheme driven by quality of solution, instead of elapsed time, and asynchronous communications between processes, instead of synchronous migration phases. However, the communication protocol and the double-ring topology proposed still compromise the efficiency of the solution. Although the communication phase in this algorithm avoids an all-to-all communication step, that would result in a bottleneck, it presents an important drawback: the delay in the communication reception in the furthest processes. When a process obtains such a good solution to be spread to the rest, it exchanges information with both its previous and its next neighbors. These neighbors will also exchange this information with their neighbors, and so on. Thus, an all-to-all communication is avoided, and the information is communicated to the rest through the double-ring. However, since each process will only exchange information during the communication step, placed at the end of each iteration, the delay encountered to transmit a new best solution from one process to their furthest neighbors in the ring could be excessive. The convergence of the cooperative algorithm is altered, since many communications would arrive delayed, thus being useless for many processes that already have reached better solutions by themselves. This chapter describes an upgraded proposal that includes a new communication protocol to avoid these issues and improve the efficiency of the asynchronous cooperative scheme. Also, despite the reduction in the convergence time obtained by the aCeSS algorithm, the diverse eSS methods running in different processors often cause situations where only some processes, the most promising ones, are able to share solutions with the rest. This limits the scalability of the proposal, since those processes that never obtain solutions to be shared, can be considered idle processes. Therefore, this chapter also elaborates on a self-adaptive procedure that allows for reconfiguring the slowest processes with the successful parameters of the promising ones.

3.3. Improving eSS through parallel cooperative searching

As already commented in previous chapters, achieving an efficient parallelization of a metaheuristic is usually a complex task since the search of new solutions depends on previous iterations of the algorithm, which not only complicates the parallelization itself but also limits the achievable speedup.

In the case of the eSS algorithm, the majority of time-consuming operations (evaluations of the cost function) are located in inner loops (lines 13-23 in Algorithm 5) which can be easily performed in parallel. However, since the main loop of the algorithm (line 7 in Algorithm 5) presents dependencies between different iterations and, furthermore, the dimension of the combination loop is rather small, a fine-grained parallelization would limit the scalability in distributed systems. Thus, a more effective solution is a coarse-grained parallelization that implies finding a parallel variant of the sequential algorithm. An island-model approach [5] can be used, so that the reference set is divided into subsets (*islands*) where the eSS is executed isolated and sparse individual exchanges are performed among islands to link different subsets. This solution drastically reduces the communications between distributed processes. However, its scalability is again heavily restrained by the small size of the reference set in the eSS method. Reducing the already small reference set by dividing it between the different islands will have a negative impact on the convergence of the eSS. Thus, building upon the ideas outlined in [219], here we propose an island-based method where each island performs an eSS using a different *RefSet*, while they cooperate modifying the systemic properties of the individual searches.

As already pointed out in Chapter 2, current HPC systems include clusters of multicore nodes that can benefit from the use of a hybrid programming model, in

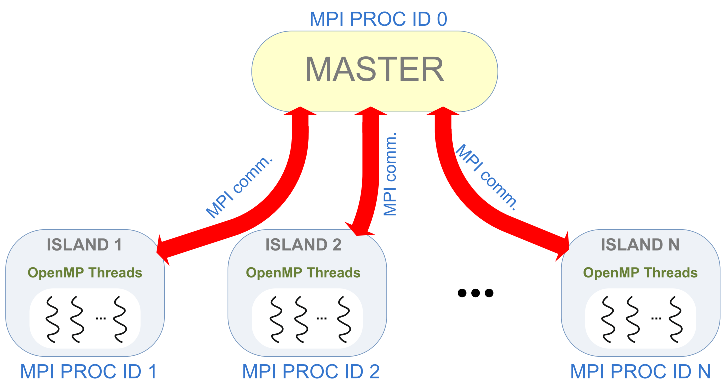


Figure 3.3: Schematic representation of the proposed hybrid MPI+OpenMP algorithm.

which a message passing library, such as MPI (Message Passing Interface), is used for the inter-node communications while a shared memory programming model, such as OpenMP, is used intra-node. Even though programming using a hybrid MPI+OpenMP model requires some effort from application developers, this model provides several advantages such as reducing the communication needs and memory consumption, as well as improving load balance and numerical convergence [106].

Thus, the combination of a coarse-grained parallelization using a distributed-memory paradigm and an underlying fine-grained parallelization of the individual tasks with a shared-memory model is an attractive solution for improving the scalability of the proposal. A hybrid implementation combining MPI and OpenMP is explored in this chapter. The proposed solution pursues the development of an efficient cooperative eSS, focused on both the acceleration of the computation by performing separate evaluations in parallel and the convergence improvement through the stimulation of the diversification in the search and the cooperation between different islands. MPI is used for communication between different islands, that is, for the cooperation itself, while OpenMP is used inside each island to accelerate the computation of the evaluations. Figure 3.3 schematically illustrates this idea, where each MPI process is an island that performs an isolated eSS. Cooperation between islands is achieved through the master process by means of message passing. Each MPI process (island) spawns multiple OpenMP threads to perform the evaluations within its population in parallel.

3.3.1. Fine-grained parallelization

The most time consuming task in the eSS algorithm is the evaluation of the solutions (cost function values corresponding to new vectors in the parameter space). This task appears in several steps of the algorithm, such as in the creation of the initial random $ndiverse$ solutions, in the combination loop to generate new solutions, and in the go-beyond method (lines 4, 13-23 in Algorithm 5, and Algorithm 6, respectively). Thus, we have decided to perform all these evaluations in parallel using the OpenMP library.

Algorithm 8 shows a basic pseudocode for performing the solutions' evaluation in parallel. As can be observed, every time an evaluation of the solutions is needed, a parallel loop is defined. In OpenMP, the execution of a parallel loop is based on the fork-join programming model. In the parallel section, the running thread creates a group of threads, so that the set of solutions to be evaluated are divided among them and each evaluation is performed in parallel. At the end of the parallel loop, the different threads are synchronized and finally joined again into only one thread. Due to this synchronization, load imbalance in the parallel loop can cause significant delays. This is the case of the evaluations in the eSS, since different evaluations can have entirely different computational loads. Thus, a dynamic schedule clause must be used so that the assignment can vary at run-time and the iterations are handed out to threads as they complete their previously assigned evaluation. Finally, at the end of the parallel region, a reduction operation allows for counting the number of total evaluations performed.

3.3.2. Coarse-grained parallelization

The coarse-grained parallelization proposed is based on the cooperation between parallel processes. For this cooperation to be efficient in large-scale difficult problems, each island must adopt a different strategy to increase the diversification in the search. The idea is to run in parallel processes with different degrees of *aggressiveness.* Some processes will focus on diversification (global search) increasing the probabilities of finding a feasible solution even in a *rough* or difficult space. Other processes will concentrate on intensification (local search) and speed-up the com-

Algorithm 8: Parallel solutions' evaluation.

```
neval = 0;
$$ parallel do (dynamic schedule, private(eval,newsol,i)
  reduction(+:neval));
for i=1 to numSolutions do
    newsol = solutions(:,i);
    eval = f_eval(newsol);
    neval ++;
end
$$ end parallel do;
```

putations in *smoother* spaces. Cooperation among them enables each process to benefit from the knowledge gathered by the rest. However, an important issue to be solved in parallel cooperative schemes is the coordination between islands so that the processes' stalls due to synchronizations are minimized in order to improve the efficiency and, specifically, the scalability of the parallel approach.

The solution proposed in this chapter follows a popular centralized master-slave approach. However, as opposed to most master-slave approaches, in the proposed solution the master process does not play the role of a central globally accessible memory. The data is completely distributed among the slaves (islands) that perform a sequential eSS each. The master process is in charge of the cooperation between the islands. The main features of the proposed scheme presented in this chapter are:

- *cooperation between islands*: by means of the exchange of information driven by the quality of the solutions obtained in each slave, rather than by elapsed time, to achieve more effective cooperation between processes.
- *asynchronous communication protocol*: to handle inter-process information exchange, avoiding idle processes while waiting for information exchanged from other processes.
- *self-adaptive procedure*: to dynamically change the settings of those slaves that do not cooperate, sending to them the settings of the most promising processes.

In the following subsections we describe in detail the implementation of the new self-adaptive cooperative enhanced Scatter Search algorithm (saCeSS), focusing on these three main features and providing evidence for each one of the design decisions taken.

Cooperation between islands

Some fundamental issues have to be addressed when designing cooperative parallel strategies [207], such as what information is exchanged, between which processes it is exchanged, when and how information is exchanged and how the imported information is used. The solution to these issues has to be carefully designed to avoid well-documented adverse impacts on diversity that may lead to premature convergence.

The cooperative search strategy proposed in this chapter accelerates the exploration of the search space through different mechanisms: launching simultaneous searches with different configurations from independent initial points and including cooperation mechanisms to share information between processes. On the one hand, a key aspect of the cooperation scheme is deciding when a solution is considered *promising* and deserves to be spread to the rest of the islands. The accumulated knowledge of the field indicates that information exchange between islands should not be too frequent to avoid premature convergence to local optima [208,209]. Thus, exchanging all current-best solutions is avoided to prevent the cooperation entries from filling up the islands' populations and leading to a rapid decrease of the diversity. Instead, a threshold is used to determine when a new best solution significantly outperforms the current-best solution and it deserves to be spread to the rest. The threshold selection adds a new degree of freedom that needs to be fixed to the cooperative scheme. The adaptive procedure described further in this section solves this issue.

On the other hand, the strategy used to select those members of the *RefSet* to be replaced with the incoming solutions, that is, with *promising* solutions from other islands, should be carefully decided. One of the most popular selection/replacement policies for incoming solutions in parallel metaheuristics is to replace the *worst* solution in the current population with the incoming solution when the value of the latter is better than that of the former. However, this policy is contrary to the

RefSet update strategy used in the eSS method, going against the idea that *parents* can only be replaced by their own *children* to avoid loss of diversity and to prevent premature stagnation. Since an incoming solution is always a promising one, replacing the *worst* solution will promote this entry to higher positions in the sorted *RefSet*. It is easy to realise that, after a few iterations receiving new *best* solutions, considering the small *RefSet* in the eSS method, the initial population in each island will be lost and the *RefSet* will be full of foreign individuals. Moreover, all the island populations would tend to be uniform, thus, losing diversity and potentially leading to rapidly converge to suboptimal solutions. Replacing the *best* solution instead of the *worst* one solves this issue most of the times. However, several members of the initial population could still be replaced by foreign solutions. Thus, the selection/replacement policy proposed in this chapter consists in labeling one member of the *RefSet* as a cooperative member, so that a foreign solution can only enter the population by replacing this cooperative solution. The first time a shared solution is received, the *worst* solution in the *RefSet* will be replaced. This solution will be labeled as a *cooperative* solution for the next iterations. A *cooperative* solution is handled like any other solution in the *RefSet*, being combined and improved following the eSS algorithm. It can also be updated by being replaced by its own offspring solutions. Restricting the replacement of foreign solutions to the *cooperative* entry, the algorithm will evolve over the initial population and still *promising* solutions from other islands may benefit the search in the next iterations.

As described before, the eSS method already includes a stagnation checking mechanism (lines 29-34 in Algorithm 5) to replace those solutions of the population that cannot be improved in a certain number of iterations of the algorithm by random generated solutions. The *nstuck* counter is used to watch out the stagnation of individuals. Diversity is automatically introduced in the eSS when the members in the *RefSet* appeared to be stuck. In the cooperative scheme this strategy may punish the *cooperative* solution by replacing it too early. In order to avoid that, a *nstuck* larger than that of other members of the RefSet is assigned to the *cooperative* solution.

Asynchronous communication protocol

An important aspect when designing the communication protocol is the interconnection topology of the different components of the parallel algorithm. A widely used topology in master-slave models, the *star* topology, is used in our proposal. It enables different components of the parallel algorithm to be tightly coupled, thus quickly spreading the solutions to improve the convergence. The master process is in the center of the star and all the rest of the processes (slaves) exchange information through the master. The distance[1] between any two slaves is always two, therefore it avoids communication delays that would harm the cooperation between processes.

The communication protocol is designed to avoid processes' stalls if messages have not arrived during an external iteration, allowing for the progress of the execution in every individual process. Both the emission and reception of the messages are performed using non-blocking operations, thus allowing for the overlap of communications and computations. This is crucial in the application of the saCeSS method to solve large-scale difficult problems since the algorithm success heavily depends on the diversification degree introduced in the different islands that would result in an asynchronous running of the processes and a computationally unbalanced scenario. Figure 3.4 illustrates this fact by comparing a synchronous cooperation scheme (CeSS [219]) with the asynchronous cooperation proposed here. In a synchronous scheme, all the processes need to be synchronized during the cooperation stage, while in the proposal, each process communicates its promising results and receives the cooperative solutions to/from the master in an asynchronous fashion, avoiding idle periods.

Self-adaptive procedure

The adaptive procedure aims to dynamically change, during the search process, several parameters that impact the success of the parallel cooperative scheme. In the proposed solution, the master process controls the long-term behavior of the parallel searches and their cooperation. An iterative life cycle model has been followed for the design and implementation of the tuning procedure and several parameter

[1]The distance between two nodes in a topology is defined by the minimum number of nodes that must be traversed to join them.

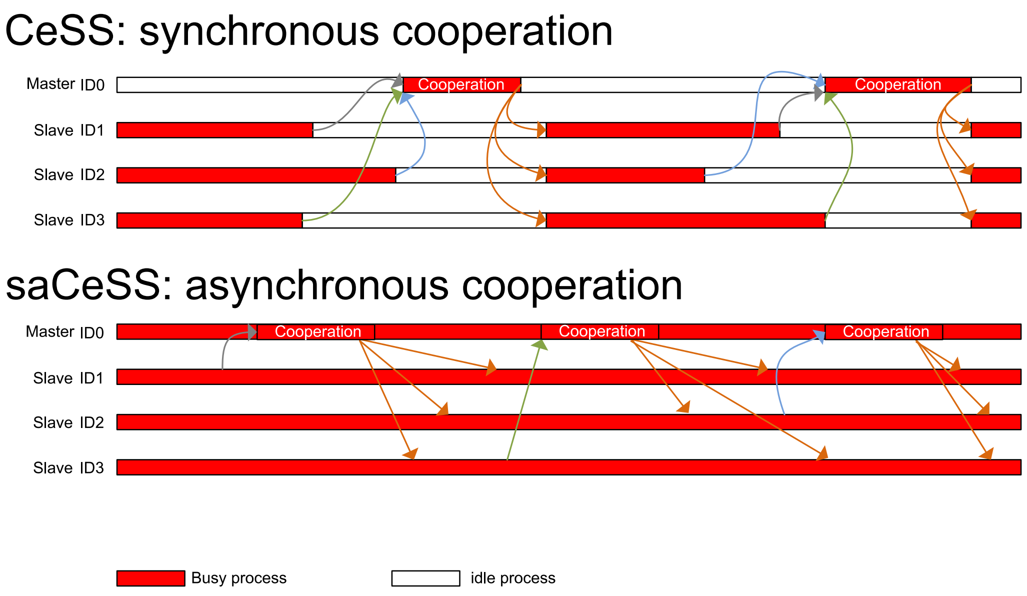


Figure 3.4: Visualization of performance analysis against time comparing synchronous versus asynchronous cooperation schemes.

estimation benchmarks have been used for the evaluation of the proposal in each iteration, in order to refine the solution to tune widespread problems.

First, the master process is in charge of the threshold selection used to decide which cooperative solutions that arrive at the master are qualified to be spread to the island. If the threshold is too large, cooperation will occur only sporadically, and its efficiency will be reduced. However, if the threshold is too small, the number of communications will increase, which not only negatively affects the efficiency of the parallel implementation, but is also often counterproductive since solutions are generally similar, and the receiver processes have no chance of actually acting on the incoming information. It has also been observed that excess cooperation may rapidly decrease the diversity of the parts of the search space explored (many islands will search in the same region) and bring an early convergence to a non-optimal solution. For illustrative purposes Figure 3.5 shows the percentage of improvement of a new best solution with respect to the previous best known solution, as a function of the number of cooperation events, when using a very low fixed threshold. Considering that at the beginning of the execution the improvements in the local solutions will be notably larger than at the end, an adaptive procedure that allows starting with a large threshold and decrease it with the search progress will improve the efficiency

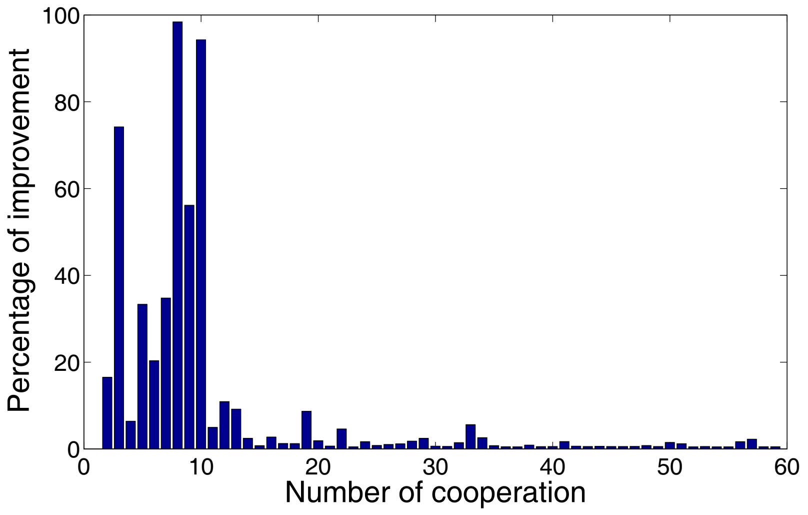


Figure 3.5: Improvement as a function of cooperation. Results obtained from benchmark B4, described in Section 3.4.

of the cooperation scheme. The suggested threshold to begin with is 10%, that is, incoming solutions that improve the best known solution in the master process by at least 10% are spread to the islands as cooperative solutions. Once the search progresses and most of the incoming solutions are below this threshold of improvement, the master reduces the threshold to one half. This procedure is repeated, so that the threshold is reduced, driven by the incoming solutions (i.e., the search progress in the islands). Note that if a excessively high threshold is selected, it will rapidly decrease to an adequate value for the problem at hand, when the master process ascertains that there are insufficient incoming solutions below this threshold.

Second, the master process is used as a scoreboard intended to dynamically tune the settings of the eSS in the different islands. As commented above, each island in the proposed scheme performs a different eSS. An *aggressive* island performs frequent local searches, trying to refine the solution very quickly and keeps a small reference set of solutions. It will perform well in problems with parameter spaces that have a smooth shape. On the other hand, *conservative* islands have a large reference set and perform local searches only sporadically. They spend more time combining parameter vectors and exploring the different regions of the parameter space. Thus, they are more appropriate for problems with rugged parameter spaces. Since the exact nature of the problem at hand is always unknown, it is recommended to choose, at the beginning of the scheme, a range of settings that yields conservative, aggressive, and intermediate islands. However, a procedure that adaptively changes the settings in the islands during the execution, favoring those settings that exhibit the highest success, will further improve the efficiency of the evolutionary search.

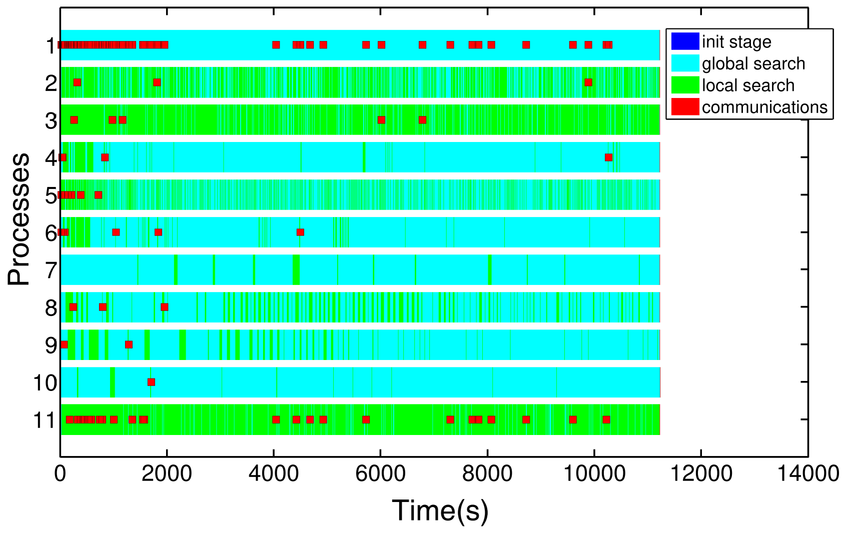
There are several configurable settings that determine the strategy (conservative/aggressive) used by the sequential eSS algorithm, and whose selection may have a great impact in the algorithm performance. Namely, these settings are:

- Number of elements in the reference set (*dimRefSet*, defined in line 1 in Algorithm 5).
- Minimum number of iterations of the eSS algorithm between two local searches (*local.n2*, line 11 in Algorithm 7).
- Balance between intensification and diversification in the selection of initial points for the local searches (*balance*, line 16 in Algorithm 7).

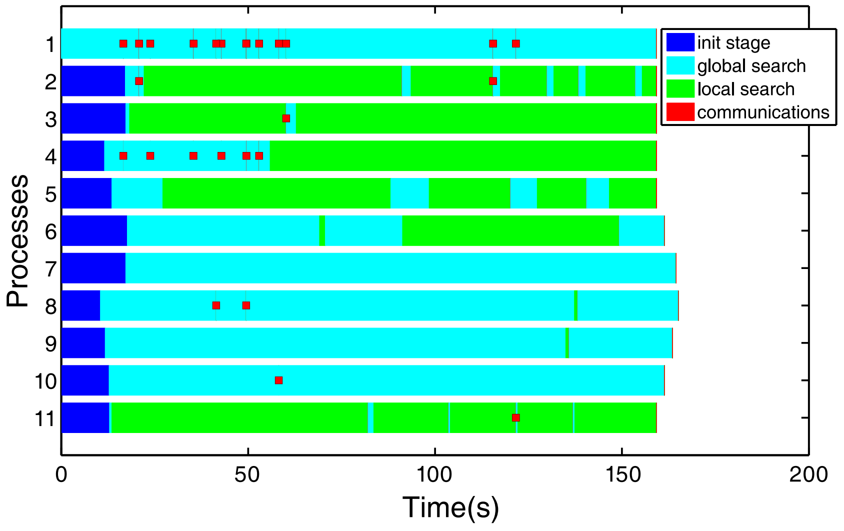
All these settings have qualitatively the same influence on the algorithm's behavior: large setting values lead to conservative executions, while small values lead to aggressive executions.

Designing a self-adaptive procedure that identifies those islands that are becoming failures and those that are successful is not an easy task. To decide which are the most promising islands, the master process serves as a scoreboard whereby the islands are ranked according to their potential. In the rating of the islands, two facts have to be considered: (1) the number of total communications received in the master from each island, to identify promising islands among those that intensively cooperates with new *good* solutions; and (2) for each island, the moment when its last solution has been received, to prioritize those islands that have more recently cooperated. A good balance between these two factors will produce a more accurate scoreboard. To better illustrate this problem Figure 3.6 shows, as an example, Gantt diagrams where the communications are colored in red. Process 1 is the master, and processes 2-11 are slaves (islands). Red dots represent asynchronous communications between master and slaves. Light blue marks represent global search steps, while green marks represent local search steps. These figures correspond to two different examples, intended to illustrate the design decisions, described in the text, taken in the self-adaptive procedure to identify successful and failure islands. At a time of $t = 2000$, process 5 has a large number of communications performed, however, all these communications were performed a considerable time ago. On the other hand, process 2 has just communicated a new solution, but presents a smaller number of total communications performed. To accurately update the scoreboard, the rate of each island is calculated in the master as the product of the number of communications performed and the time elapsed from the beginning of the execution until the last reception from that island. In the example above, process 6 achieves a higher rate because it presents a better balance between the number of communications and the time elapsed since the last reception.

Identifying the *worst* islands is also cumbersome. Those processes at the bottom of the scoreboard are there because they do not communicate sufficient solutions or because a considerable amount of time has passed since their last communication. However, they can be either non-cooperating (less promising) islands or more aggressive ones. An aggressive thread often calls the local solver, performing longer iterations, and thus being unable to communicate results as often as conservative islands can do so. To better illustrate this problem, Figure 3.6(b) shows a new Gantt diagram. At a time of $t = 60$, process 4 will be at the top of the scoreboard because it often obtains promising solutions. This is a conservative island. Process 3 will be at the bottom of the scoreboard because at that time it has not yet communicated a significant result. The reason is that process 3 is an aggressive slave that is still performing its first local search. To accurately identify the non-cooperating islands, the master process would need additional information from islands that would imply extra messages in each iteration of the eSS. The solution implemented is that each island decides by itself whether it is evolving in a promising mode or not. If an island detects that it is receiving cooperative solutions from the master but it cannot improve its results, it will send the master a reconfiguration request. The master, will then communicate to this island the settings of the island on the top of the scoreboard.



(a) Gantt diagram 1



(b) Gantt diagram 2

Figure 3.6: Gantt diagrams representing the tasks and cooperation between islands against execution time.

3.3.3. Comprehensive overview of the saCeSS method

The pseudocode for the master process in saCeSS method is shown in Algorithm 9, while the basic pseudocode for each slave is shown in Algorithm 10. At the beginning of the algorithm, a local variable present in the master and in each slave is declared to keep track of the best solution shared in the cooperation step. The master process sets the initial threshold, initiates the scoreboard to keep track of the cooperation rate of each slave, and begins to listen to the requests and cooperations arriving from the slaves. Each time the master receives from a slave a solution that significantly improves the current best known solution (*BestKnownSol*), it increments the score of this slave on the board.

Each slave creates its own population matrix of *ndiverse* solutions. Then an initial *RefSet* is generated for each process with *dimRefSet* solutions with the best elements and random elements. Again, different *dimRefSet* are possible for different processes. The rest of the operations are performed within each *RefSet* in each process, in the same way as in the serial eSS implementation. Every external iteration of the algorithm, a cooperation phase is performed to exchange information with the rest of the processes in the parallel application. Whenever a process reaches the cooperation phase, it checks if any message with a new best solution from the master has arrived at its reception memory buffer. If a new solution has arrived, the process checks whether this new solution improves the current best solution (*BestKnownSol*) or not. If the new solution improves the current one, the new solution promotes to be the *BestKnownSol*. The loop to check the reception of new solutions must be repeated until there are no more shared solutions to attend. This is because the execution time of one external iteration may be very different from one process to another, due to the diversification strategy explained before. Thus, while a process has completed only one external iteration, their neighbors may have completed more and several messages from the master may be waiting in the reception buffer. Then, the *BestKnownSol* has to replace the cooperation entry in the process *RefSet*.

After the reception step, the slave process checks whether its best solution improves in, at least, an ϵ the *BestKnownSol*. If this is the case, it updates *BestKnownSol* with its best solution and sends it to the master. Note that the ϵ used in the slaves is not the same as the ϵ used in the master process. The slaves use a

Algorithm 9: saCeSS algorithm. Pseudocode for the *master* process.

```
BestKnownSol = DBL_MAX;
! Initial threshold
$\epsilon$=0.1;
nRefuse=0;
! Scoreboard information
slaveComm(:)=0;
slaveScore(:)=0;
for i=1 to nslaves do
    scoreBoard(i)=i;
end
repeat
    ! Cooperation step
    recvflag=true;
    sendflag=false;
    while recvflag do
        Non_Blocking_Recv(RecvSol,slave,recvflag);
        if $((BestKnownSol - RecvSol)/BestKnownSol) < \epsilon$ then
            BestKnownSol=RecvSol;
            sendflag=true;
            ! Score slave
            slaveComm(slave)++;
            slaveScore(slave)=slaveComm(slave)*elapsedTime();
        else
            nRefuse++;
        end
    end
    if sendflag then
        Non_Blocking_Send(BestKnownSol,slaves);
    end
    ! Adapt threshold considering refused solutions
    if $nRefuse > nslaves$ then
        $\epsilon = \epsilon/2$ ;
        nRefuse=0;
    end
    ! Adapt slaves' settings using scoreboard
    recvflag=true;
    while recvflag do
        Non_Blocking_Recv(Request,slave,recvflag);
        Sort(scoreBoard);
        Non_Blocking_Send(NewSettings[scoreBoard(0)],slave);
    end
until stopping criterion;
```

Algorithm 10: saCeSS algorithm. Pseudocode for the *slave* processes

```
1  BestKnownSol = DBL_MAX;
2  recvSolutions=0;
3  iter_solver=0;
4  $N_{eval}$=0;
5  Create_Population(ndiverse);
6  Generate_RefSet(RefSet, dimRefSet);
7  repeat
8      ! Serial eSS: (1) RefSet order and duplicity check, (2) Solution combination,
        (3) RefSet update, (4) Extra mechanisms and (5) Improvement method (see
        Algorithms 5, 6 and 7).
9      ! Cooperation step
10     sendflag=false;
11     recvflag=true;
12     replaceflag=false;
13     while recvflag do
14         Non_Blocking_Recv(RecvSol,master,recvflag);
15         if $RecvSol < BestKnownSol$ then
16             BestKnownSol=RecvSol;
17             recvSolutions++;
18             replaceflag=true;
19         end
20     end
21     if replaceflag then
22         Replace_Solution(BestKnownSol);
23     end
24     if $((BestKnownSol - bestSol)/bestSol) < \epsilon$ then
25         BestKnownSol=bestSol;
26         sendflag=true;
27     end
28     if sendflag then
29         Non_Blocking_Send(BestKnownSol,master);
30         $N_{eval}$=0;
31         recvSolutions=0;
32     end
33     ! Adaptive step
34     if $recvSolutions > (10 \times sendSolutions) + 20$ OR $N_{eval} > (N_{par} \times 5000)$ then
35         Non_Blocking_Send(Request,master);
36     end
37     Non_Blocking_Recv(NewSettings,master,recvflag);
38 until stopping criterion;
```

rather small ϵ so that many *good* solutions will be sent to the master. The master, in turn, is in charge of selecting those incoming solutions that are qualified to be spread, thus, its ϵ begins with quite a large value that decreases when the number of refused solutions increases and no incoming solution overcomes the current ϵ.

Finally, before the end of the iteration, the adaptive phase is performed. Each slave decides if it is progressing in the search by checking if:

$$N_{eval} > N_{par} \times 5000$$

where N_{eval} is the number of evaluations performed by this process since its last cooperation with the master and N_{par} is the number of parameters of the problem. Note that, in general, the larger the number of parameters to be estimated, the harder the problem is. Thus, the reconfiguration condition depends on the problem at hand. Besides, if the number of received solutions is greater than the number of solutions sent, that is, if other processes are cooperating much more than itself, the reconfiguration condition is also met. Summarizing, if a process detects that it is not improving while it is receiving solutions from the master, it sends a request for reconfiguration to the master process. The master listens to these requests and sends to those slaves the settings of the most promising ones, i.e., those that are on the top of the scoreboard.

Finally, the saCeSS algorithm repeats the external loop until the stopping criterion is met. The current version can consider three different stopping criteria (or any combination among them): maximum number of evaluations, maximum execution time and a *value-to-reach (VTR)*. While the VTR is usually known in benchmark problems (such as those used below), for a new problem, the VTR will be, in general, unknown. Thus, in more realistic cases, the recommended practice in metaheuristics is to perform some trial runs and then analyze the convergence curves in order to find sensible values for the maximum number of evaluations and/or execution time.

For illustrative purposes, Figure 3.7 graphically shows an example of the execution of the saCeSS method. Note that different processes are executing a different eSS. Since they run asynchronously, they might be in different stages at every time moment. Thus, cooperation between these different searches should also be performed in an asynchronous fashion, avoiding stalls if any of the islands is involved

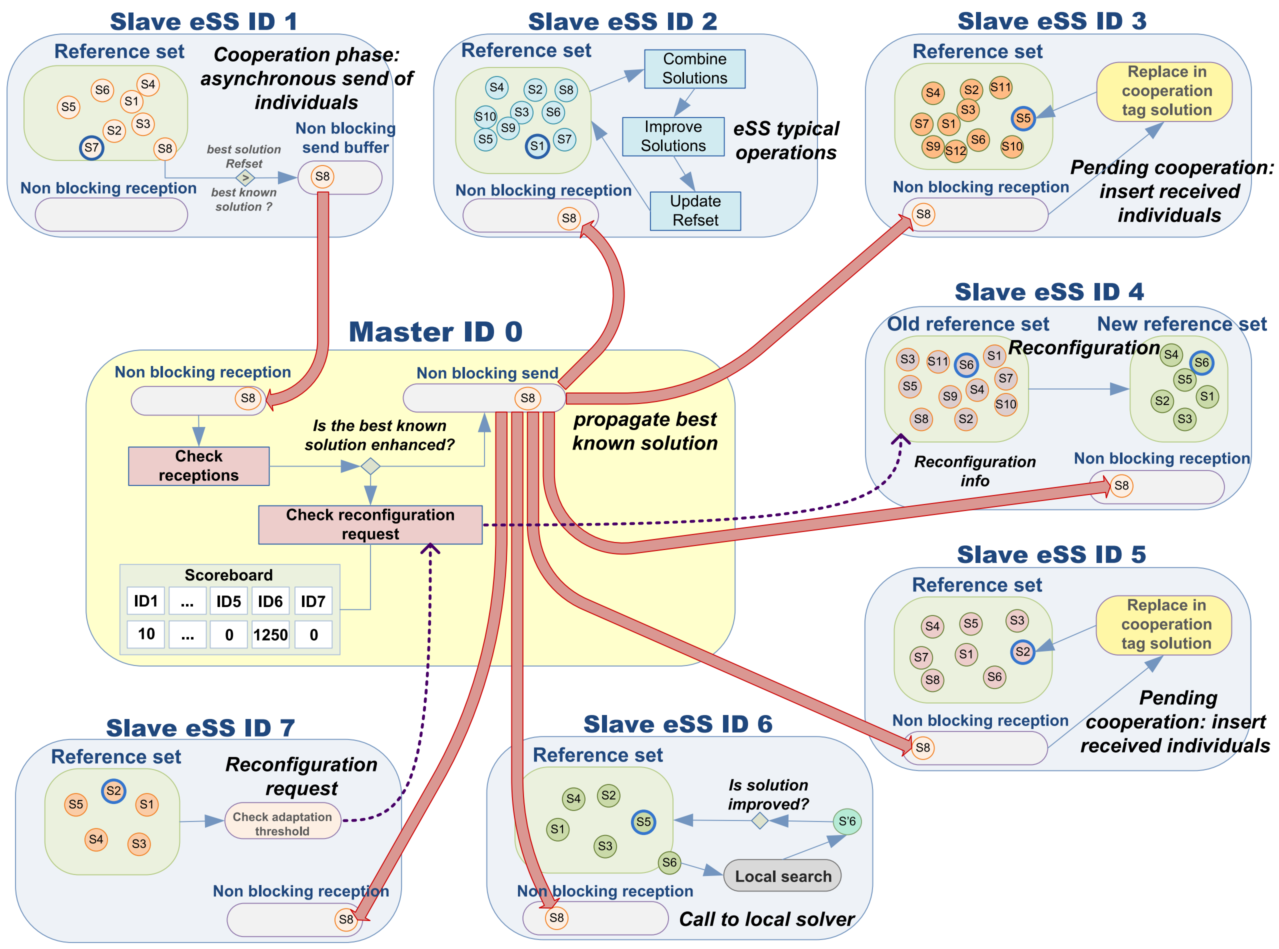


Figure 3.7: saCeSS: an illustrative representation of the different mechanisms of communication and adaptation proposed

in a time consuming phase, such as the execution of the local solver (see process ID 6 in the figure), while other islands (processes ID 1, ID 3 and ID 5 in the figure) are in the cooperation phase. When an island cannot attend to a cooperation reception, the message will be stored in the process as a pending cooperation, avoiding the blocking of the sender process (see processes ID 3 or ID 5 in the figure, attending pending cooperations). The master process (ID 0 in the figure) is in charge of the cooperation between parallel searches and their long-term behavior. It maintains a scoreboard to guide the adaptive procedure that tunes the settings of the eSS in the different islands. When an island detects that it is not progressing in its search (see process ID 7 in the figure), it sends the master a reconfiguration request. The master communicates, to those islands that request a reconfiguration, the settings of the most promising searches according to its scoreboard (see process ID 4 in the figure).

3.4. Experimental results

The proposed saCeSS method has been applied to a set of benchmarks from the BioPreDyn-bench suite [220], with the goal of assessing its efficiency in challenging parameter estimation problems in computational system biology:

- *Problem B1*: genome-wide kinetic model of *S. cerevisiae.* It contains 276 dynamic states, 44 observed states and 1759 parameters.
- *Problem B2*: dynamic model of the central carbon metabolism of *E. coli.* It consists of 18 dynamic states, 9 observed states and 116 estimable parameters.
- *Problem B3*: dynamic model of enzymatic and transcriptional regulation of the central carbon metabolism of *E. coli.* It contains 47 dynamic states (fully observed) and 178 parameters to be estimated.
- *Problem B4*: kinetic metabolic model of Chinese Hamster Ovary (CHO) cells, with 34 dynamic states, 13 observed states and 117 parameters.
- *Problem B5*: signal transduction logic model, with 26 dynamic states, 6 observed states and 86 parameters.

- *Problem B6*: dynamic model describing the gap gene regulatory network of the vinegar fly, *Drosophila melanogaster*. It consists of three processes formalized with 108-212 ODEs, and resulting in a model with 37 unknown parameters.

Different experiments have been conducted using the cooperative methods, and its performance has been compared with other different parallel versions of the eSS, namely: an embarrassingly-parallel non-cooperative enhanced Scatter Search (*np*-eSS), a previous cooperative synchronous version (CeSS) described in [219], and our previous asynchronous cooperative scheme (aCeSS) proposed in [166], that has been already described in the Related Work of this chapter.

The *np*-eSS algorithm consists of *np* independent eSS runs (being *np* the number of available processors) performed in parallel without cooperation between them and reporting the best execution time of the *np* runs. Diversity is introduced in these *np* eSS runs, alike in the cooperative methods, that is, each one executes a separate eSS using different strategies.

It should be noted that the eSS [60] and CeSS [219] methods have been originally implemented in Matlab. Both algorithms have been now coded in F90 to perform an honest comparison with the proposed saCeSS method. For the same reason, the CeSS method has been re-written using the MPI library for the communications between master and slaves, since its original reported implementation used *jPar* [109].

The experiments reported here have been performed in the cluster Pluton, a multicore cluster with 16 nodes powered by two octa-core Intel Xeon E5-2660 CPUs with 64 GB of RAM. The cluster nodes were connected through an InfiniBand FDR network. For problem *B3* this cluster could not be used because its execution exceeds the maximum allowed job length. Thus, the cluster of the Bioprocess Engineering Group at IIM-CSIC that consists of 4 nodes powered by two quadcore Intel Xeon E5420 CPUs with 16 GB of RAM and 8 nodes powered by two quadcore Intel Xeon E5520 CPUs with 24 GB of RAM, connected through a Gigabit Ethernet network, has been used for this problem.

The computational results shown in this chapter were analyzed from a horizontal view [87], that is, assessing the performance by measuring the time needed to reach a given target value. To evaluate the efficiency of the proposal, experiments with a stopping criteria based on a *value-to-reach* (VTR) were performed. The VTR used was the optimal fitness value reported in [220]. Also, since comparing different metaheuristics is not an easy task, due to the substantial dispersion of computational results due to the stochastic nature of these methods, each experiment reported in this section was performed 20 times and a statistical study was carried out.

3.4.1. Performance evaluation of the coarse-grained parallelization and the self-adaptive mechanism

The cooperation between processes in the coarse-grained parallelization can modify the systemic properties of the eSS algorithm and therefore its macroscopic behavior. The same happens with the self-adaptive mechanism proposed. Thus, the first set of experiments shown in this section disables the fine-grained parallelization, since it does not alter the convergence properties of the algorithm, to evaluate solely the impact of the coarse-grained parallelization, as well as the self-adaptive mechanism.

Table 3.1 displays, for each benchmark and each method, the number of external iterations performed (line 7 in Algorithm 5), the average number of evaluations needed to achieve the VTR, the mean and standard deviation execution time of all the runs in the experiments, and the speedup achieved. The compared methods are: *np*-eSS, CeSS, aCeSS, and the proposed saCeSS method. The speedup was calculated versus the sequential eSS, except for problem B3, whose prohibitively large execution times did not allow us to complete the set of sequential eSS tests, and forced us to reduce the rest of experiments. Therefore, for problem B3 the speedup was computed versus the 10-eSS execution time. To better evaluate the efficiency of the self-adaptive procedure proposed, Table 3.1 displays results for two kinds of experiments with the saCeSS method: ones where the self-adaptive procedure was disabled, labeled as saCeSS(non-adaptive) in the table, that is, the settings of the eSS in the different islands were not dynamically tuned during the execution; and others where this procedure was enabled, labeled as saCeSS in the table, allowing for the adaptive reconfiguration of the islands. These results were obtained using 10 processors and using the following stopping criteria: $VTR_{B1} = 1.3753 \times 10^4$, $VTR_{B2} = 2.50 \times 10^2$, $VTR_{B3} = 3.7 \times 10^{-1}$, $VTR_{B4} = 55$, $VTR_{B5} = 4.2 \times 10^3$,

Table 3.1: Performance of the coarse-grained parallelization and the self-adaptive scheme proposed.

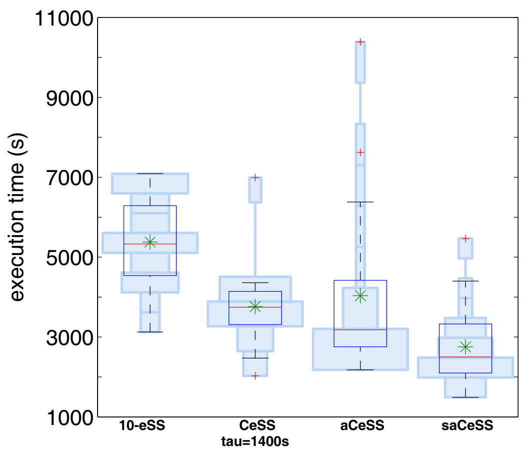
	method	**iter±std**	**evals±std**	**time±std(s)**	**sp**
	eSS	67 ± 34	43484 ± 21852	15572 ± 11310	-
	10-eSS	80±30	199214±44836	5378 ± 1070	2.9
	CeSS ($\tau = 700s$)	109±49	188131±82834	6487 ± 3226	2.4
B1	CeSS ($\tau = 1400s$)	122±41	175331±98255	5018 ± 1477	3.1
	aCeSS	107 ± 53	147946 ± 81316	4034 ± 2109	3.8
	saCeSS($_{\text{non-adap}}$)	92±35	143145±61828	3759 ± 976	4.1
	saCeSS	62±21	92122±35058	2753 ± 955	5.7
	eSS	599 ± 252	779062 ± 326592	10344 ± 5675	-
	10-eSS	450±167	1504503±541257	1914 ± 714	5.4
	CeSS ($\tau = 400s$)	452±278	1637125±1016688	2459 ± 2705	4.2
B2	CeSS ($\tau = 800s$)	508±205	1802917±690613	1911 ± 1103	5.4
	aCeSS	421 ± 278	1311108 ± 849641	1842 ± 1348	5.6
	saCeSS($_{\text{non-adap}}$)	440±192	1528793±647677	1918 ± 833	5.4
	saCeSS	846±982	1247699±1222378	1694 ± 1677	6.1
	10-eSS	10062±2528	66915128±15623835	511166 ± 135988	-
	CeSS($\tau = 50000s$)	7288±5551	52592578±35513874	332721 ± 245829	1.5*
B3	saCeSS($_{\text{non-adap}}$)	4323±3251	32604331±23357322	251305 ± 209082	2.0*
	saCeSS	4113±3130	27647470±21488783	229888 ± 238970	2.2*
	eSS	7457 ± 5492	11710828 ± 8148891	39257 ± 27364	-
	10-eSS	99±121	2230089±2068300	750 ± 692	52.3
	CeSS ($\tau = 100s$)	140±386	1665954±2921838	817 ± 1909	48.0
B4	CeSS ($\tau = 200s$)	119±87	1649723±1024833	518 ± 428	75.7
	aCeSS	76 ± 227	1351071 ± 2424864	429 ± 781	91.5
	saCeSS($_{\text{non-adap}}$)	39±30	1163458±927751	402 ± 303	97.6
	saCeSS	35±24	1017956±728328	343 ± 240	114.4
	eSS	12 ± 5	13762 ± 5499	1874 ± 753	-
	10-eSS	16±4	69448±14570	901 ± 197	2.0
	CeSS ($\tau = 200s$)	11±4	108481±36190	1481 ± 634	1.2
B5	CeSS ($\tau = 400s$)	14±3	94963±20172	996 ± 264	1.8
	aCeSS	10 ± 3	47364 ± 10955	603 ± 154	3.1
	saCeSS($_{\text{non-adap}}$)	10±2	49622±9530	637 ± 131	2.9
	saCeSS	10±3	51076±12696	658 ± 174	2.8
	eSS	5654±5208	2396490 ± 2188348	20436 ± 18705	-
	10-eSS	4659±3742	9783720±8755231	8217 ± 7536	2.4
	CeSS ($\tau = 1000s$)	5919±5079	10475485±8978383	8109 ± 7441	2.5
B6	CeSS ($\tau = 2000s$)	6108±6850	10778260±12157617	7878 ± 9400	2.6
	aCeSS	4501 ± 4485	7130396 ± 7123362	5838 ± 5859	3.5
	saCeSS($_{\text{non-adap}}$)	2501±1517	4394243±2689489	3638 ± 2302	5.6
	saCeSS	1500±1265	2594741±2214235	2177 ± 1933	9.3

* These speedup results are calculated versus *np*-eSS.

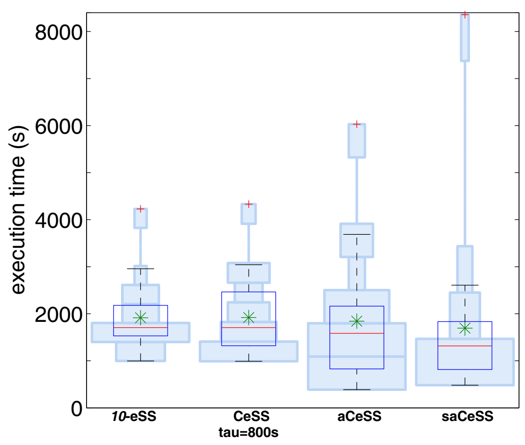
$VTR_{B6} = 1.0833 \times 10^5$.

Note that in these experiments the computational load is not shared among processors. Thus, the speedup depends on the impact that the cooperation among processes produces on achieving a good result. The results of the saCeSS (non-adaptive) show that the cooperation mechanisms and the asynchronous communication protocol proposed in the saCeSS method improve the results obtained by *10*-eSS and CeSS, both in terms of number of evaluations and execution time. As regards the CeSS method, the results can be explained both because the synchronization slows down the processes, since it implies more processes' stalls while waiting for data, and because of the effectiveness of the information exchanged. The aCeSS method outperforms the *10*-eSS and CeSS algorithms thanks to the cooperation and the asynchronous protocol. However, the delay in the communication reception caused by the ring topology is a limitation for the performance results, being clearly overcome by saCeSS for the most complex problems. The results of the saCeSS show that the self-adaptive approach improves the previous results even more. The main goal of this approach is to reduce the impact that the initial choice of the configurable settings may have in the evolution of the method. For instance, one issue of the CeSS algorithm is the selection of the migration time (τ), that is, the time between information sharing. On the one hand, this time has to be long enough to allow each of the threads to exploit the eSS capabilities. On the other hand, if the time is too long, cooperation will occur only sporadically, reducing its efficiency. On the contrary, in the saCeSS method proposed, the initial selection of the threshold to spread a cooperative solution, as well as the selection of the other configurable settings of the eSS method, will change adaptively during the execution progress.

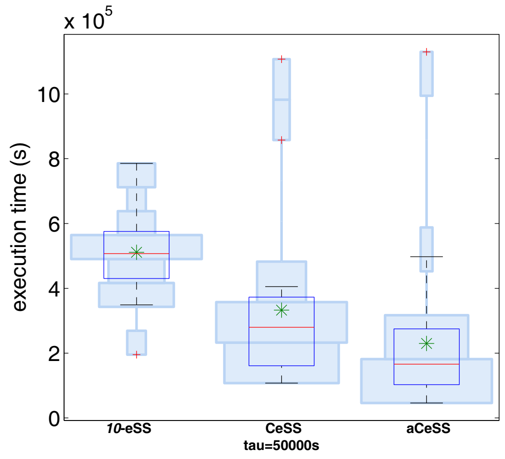
When dealing with stochastic optimization solvers, it is important to evaluate the dispersion of the computational results. Figure 3.8 illustrates how the proposed saCeSS reduces the variability of execution time in the non-cooperative *10*-eSS method. The green asterisks correspond to the mean and light blue boxes illustrate the distribution of the results. Each box with a strong blue contour represents a typical boxplot: the central red line is the median, the edges of the box are the 25th and 75th percentiles, and outliers are plotted with red crosses. This is an important feature of the saCeSS, because it reduces the average execution time for each benchmark. For instance, even in the B2 problem, where the mean execution time



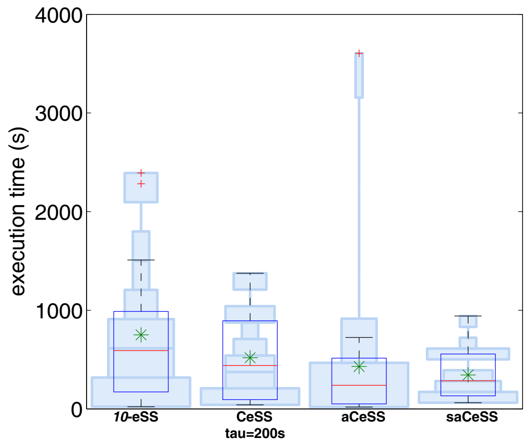
(a) B1



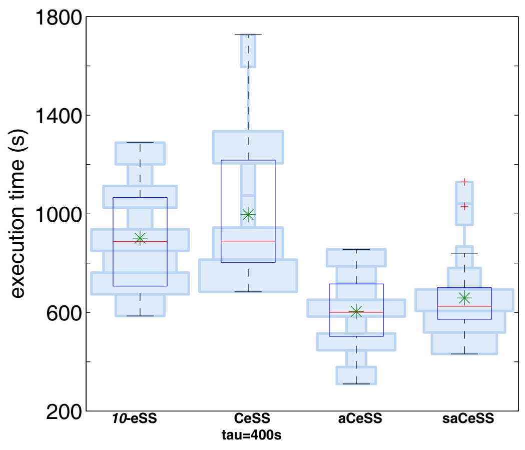
(b) B2



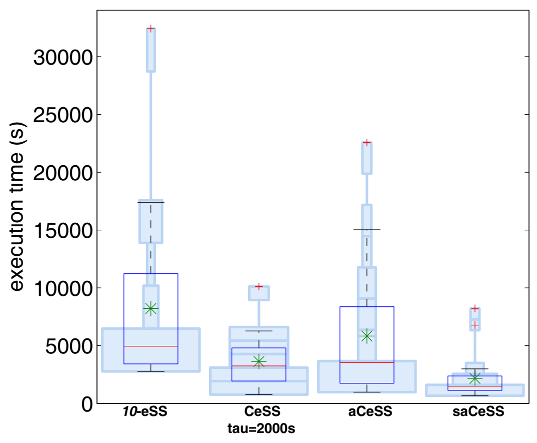
(c) B3



(d) B4



(e) B5



(f) B6

Figure 3.8: Hybrid violin/box plots of the execution times using 10 processes.

Table 3.2: p-values of the pairwise comparisons provided by the Dunn's test.

	B1	B2	B3	B4	B5	B6
saCeSS vs CeSS	0.0000	0.0630	0.0380	0.1924	0.0000	0.0000
saCeSS vs *10*-eSS	0.0000	0.0115	0.0000	0.0324	0.0001	0.0000
CeSS vs *10*-eSS	0.1850	0.2289	0.0006	0.1641	0.2128	0.3964

Table 3.3: Group classification of the optimization methods at 95% confidence level.

	B1	B2*	B3	B4	B5	B6
saCeSS	A	A	A	A	A	A
CeSS	B	A B	B	A B	B	B
10-eSS	B	B	C	B	B	B

*For problem B2, the classification at 90% confidence interval is: saCeSS(A), CeSS(B), *10*-eSS(B)

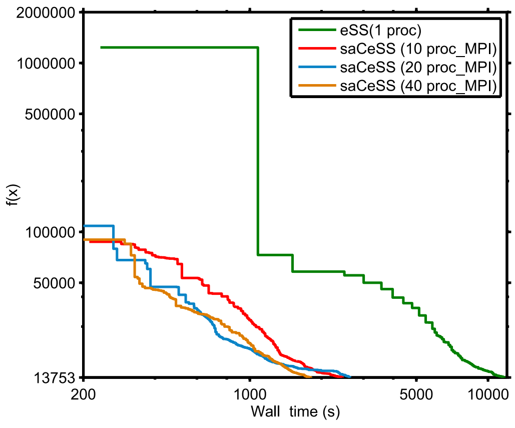
is similar in the non-cooperative and the cooperative executions, the dispersion of the results is reduced in the cooperative case.

Finally, to prove the significance of the results, a non-parametric statistical analysis has been applied to the final runtimes of the experiments over each test problem. Recent studies show that the most appropriate methods to compare the performance of different metaheuristics are the nonparametric procedures [55]. In this chapter we have applied the Kruskal-Wallis test [118] followed by Dunn's test [58], that reports the results among multiple pairwise comparisons after the Kruskall-Wallis test. The objective is to explain if the observed differences among final runtimes for each problem are due to the optimization method used (i.e., *np*-eSS, CeSS and saCeSS) or to pure randomness. Table 3.2 shows, for each problem, the p-values of Dunn's test for every pair comparison after the Kruskall-Wallis test application. Table 3.3 shows, for each problem, the classification of methods in groups at 95% confidence level according to the p-values shown above. As shown in Tables 3.2 and 3.3, the statistical results reveal that saCeSS shows better performance than *10*-eSS and CeSS in the problems considered. It is clear for problems B1, B3, B5 and B6. For problems B2 and especially B4, the differences are not as significant as in the other problems. For B2, the reason is the outlier in one of the runs with saCeSS in problem B2, which damages the results since it is the worst value among all runs and methods for this problem. For B4, the reason is the high dispersion of the results, together with the fact that many experiments achieved the convergence in the first iterations of the algorithm (note the swelling at the bottom of the violin/box plots in Figure 3.8), before the cooperation turned into effective, or the self-adaptive procedure became operational, thus, reducing the difference among the three algorithms.

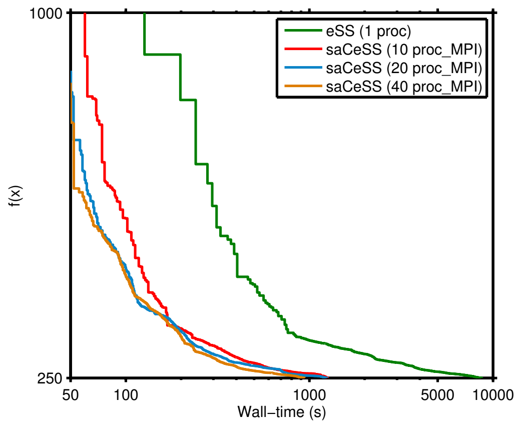
A known issue both for CeSS and aCeSS methods is their poor scalability. To assess the scalability of the coarse-grained parallelization proposed in saCeSS, experiments were carried out using 1, 10, 20 and 40 processors. The convergence curves are shown in Figure 3.9. This curves represent the logarithm of the objective function value against the execution time for those experiments that fall in the median values of the results distribution. As can be observed, when the number of processors grows, the saCeSS method keeps on improving the convergence results. This is due to the proposed asynchronous communication protocol. However, as the number of cooperative processes increases, the improvement in the algorithm performance is restrained. This is readily justified, because the self-adaptive mechanism will drive the different processes from different initial parameters to those settings that obtain successful results, which, in the long term, means that having a larger number of processes does not aim to a larger diversity and better results. Thus, the hybrid MPI+OpenMP proposal discussed in the next section aims to improve the performance results when the number of processors increases, by combining the MPI stimulation on diversification in the search with the OpenMP intensification.

3.4.2. Performance evaluation of the hybrid MPI+OpenMP proposal

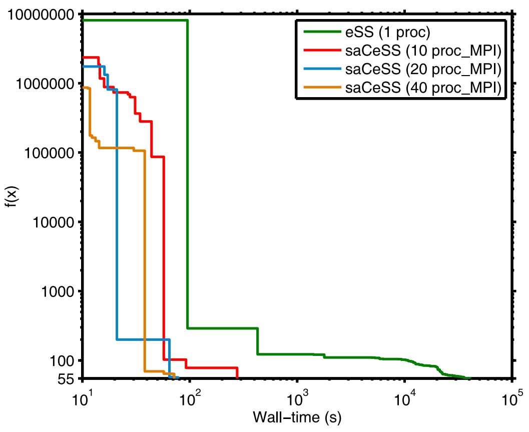
The goal of the fine-grained parallelization considered in this section is to perform the evaluation of the obtained solutions in parallel threads, thus, accelerating the execution without altering the properties of the algorithm. As already commented in previous sections, the scalability of the fine-grained parallelization is limited in the eSS algorithm, due to the small *dimRefSet*. Also, the workload is uneven, since different evaluations lead some threads to be busy for longer times. Thus, the dynamic schedule used allows the threads with small workloads to go after other



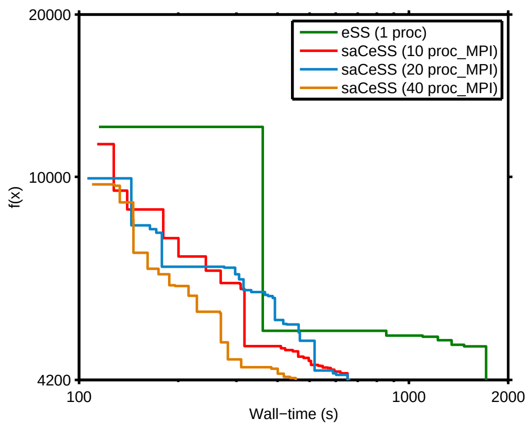
(a) B1



(b) B2



(c) B4



(d) B5

Figure 3.9: Scalability of saCeSS using 1, 10, 20 and 40 processors.

chunks of work, and hopefully, balance the work between threads. But it introduces a large overhead at runtime, as work has to be taken off from a queue.

Table 3.4 shows the performance and scalability of the hybrid MPI+OpenMP implementation proposed in this chapter. It shows the execution time and speedup results for experiments using the following stopping criteria: $VTR_{B1} = 1.3753 \times 10^4$, $VTR_{B2} = 2.50 \times 10^2$, $VTR_{B4} = 55$, $VTR_{B5} = 4.2 \times 10^3$. Benchmarks B3 and B6 were excluded of these evaluations: B3 due to our lack of available resources to

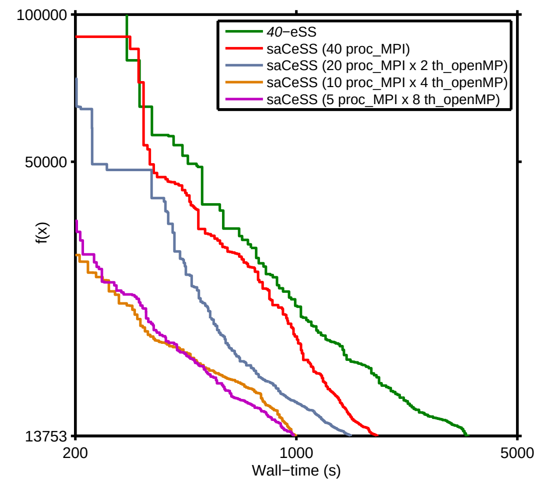
Table 3.4: Performance of the hybrid MPI+OpenMP version of saCeSS.

	meth.	**config.**	**iter±std**	**evals±std**	**time±std(s)**	**sp**
		MPIxOpenMP				
	eSS	1	67 ± 34	43484 ± 21852	15572 ± 11310	-
	saCeSS	10 (10 x 1)	62±21	92122±35058	2753 ± 955	5.6
	saCeSS	10 (5 x 2)	76±40	88186±46200	2762 ± 1273	5.6
	saCeSS	20 (20 x 1)	79±38	218871±107769	3125 ± 1349	4.9
B1	saCeSS	20 (10 x 2)	86±43	147268±75752	2299 ± 1199	6.7
	saCeSS	20 (5 x 4)	54±19	68907±27132	1288 ± 380	12.0
	saCeSS	40 (40 x 1)	51±19	274628±115644	2070 ± 650	7.5
	saCeSS	40 (20 x 2)	64±27	205766±93753	1743 ± 759	8.9
	saCeSS	40 (10 x 4)	58±30	111804±61212	1130 ± 535	13.7
	saCeSS	40 (5 x 8)	57±29	82414±38357	1078 ± 494	14.4
	eSS	1	599 ± 252	779062 ± 326592	10344 ± 5675	-
	saCeSS	10 (10 x 1)	846±982	1247699±1222378	1694 ± 1677	6.1
	saCeSS	10 (5 x 2)	630±275	916166±357665	1298 ± 700	7.9
	saCeSS	20 (20 x 1)	649±287	1901601±747379	1345 ± 619	7.6
B2	saCeSS	20 (10 x 2)	904±646	1388211±854461	962 ± 616	10.7
	saCeSS	20 (5 x 4)	609±244	925076±353594	734 ± 320	14.0
	saCeSS	40 (40 x 1)	571±542	3286363±2593258	1326 ± 1764	7.8
	saCeSS	40 (20 x 2)	766±863	2185567±2228163	951 ± 1468	10.8
	saCeSS	40 (10 x 4)	756±243	1309744±446419	506 ± 180	20.4
	saCeSS	40 (5 x 8)	522±290	801661±380553	353 ± 162	29.3
	eSS	1	7457±5492	11710828±8148891	39257±27364	-
	saCeSS	10 (10 x 1)	35±24	1017956±728328	343±240	114.4
	saCeSS	10 (5 x 2)	177±269	1854102±2331070	916±1210	42.8
	saCeSS	20 (20 x 1)	11±8	819845±548341	111±93	353.6
B4	saCeSS	20 (10 x 2)	30±23	898459±657350	215± 169	185.5
	saCeSS	20 (5 x 4)	294±692	2090044±3980316	736± 1453	53.3
	saCeSS	40 (40 x 1)	9±5	1254545±603925	78± 51	503.3
	saCeSS	40 (20 x 2)	26±26	1523817±1267964	159± 160	246.8
	saCeSS	40 (10 x 4)	37±50	878008±1178953	168± 238	233.6
	saCeSS	40 (5 x 8)	159±692	1666511±3980316	490±446	80.1
	eSS	1	12 ± 5	13762 ± 5499	1874 ± 753	-
	saCeSS	10 (10 x 1)	10±3	51076±12696	658±174	2.8
	saCeSS	10 (5 x 2)	11±3	36090±8761	520±136	3.6
	saCeSS	20 (20 x 1)	10±2	99995±21070	654±151	2.8
B5	saCeSS	20 (10 x 2)	9±2	52419±12040	368±92	5.0
	saCeSS	20 (5 x 4)	10±3	38427±10523	331± 97	5.6
	saCeSS	40 (40 x 1)	8±2	161687±40392	514±138	3.6
	saCeSS	40 (20 x 2)	9±2	100315±17758	350±68	5.3
	saCeSS	40 (10 x 4)	11±2	64594±13173	280±61	6.7
	saCeSS	40 (5 x 8)	11±3	45854±11607	248±67	7.5

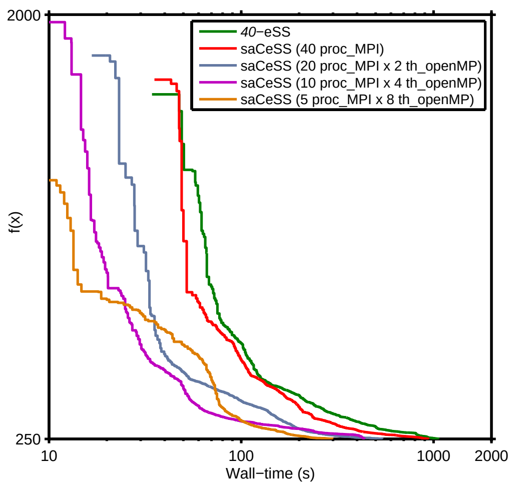
run such long executions, and B6 since its currently available implementation could not be parallelized using the OpenMP library. In most of the cases, the hybrid configurations obtain better performance than other configurations that only use MPI processes. Note that the column labeled *config.* shows the total number of cores used along with the number of MPI processes and openMP threads running in each configuration. In general, results show that, for the same number of cores, those hybrid configurations that achieve a good balance between intensification and diversification perform more effectively. For instance, the configuration of 5 MPI processes with 4 OpenMP threads each, achieves, in general, better speedup than the configuration with 10 MPI processes with 2 OpenMP threads each, while both use 20 cores. The exception is benchmark B4, whose performance is heavily affected by the number of different processes cooperating. That is, benchmark B4 greatly benefits from the diversity introduced with the scalability in the number of MPI processes versus the intensification of the OpenMP search. Thus, for benchmark B4, configurations using all the available cores to run MPI cooperative processes perform better.

Figure 3.10 shows, the convergence curves for those experiments that fall in the median values of the results distribution. The performance of *40*-eSS, that is 40 individual, non cooperative processes, is compared to the performance of saCeSS method with different hybrid configurations using 40 processors in all of them. Figure 3.11 illustrates the same comparative but from the dispersion point of view through combination of violinplots and boxplots. As has already been pointed out, when the number of available processors increases, hybrid MPI+OpenMP configurations will improve the performance versus a solely coarse-grained parallelization. This is an important result because it allows to predict a good performance of this method on currently popular HPC systems, built as clusters of multicore nodes, where MPI processes would be located in different nodes, improving diversification, while, within each node, the OpenMP threads would intensify the search.

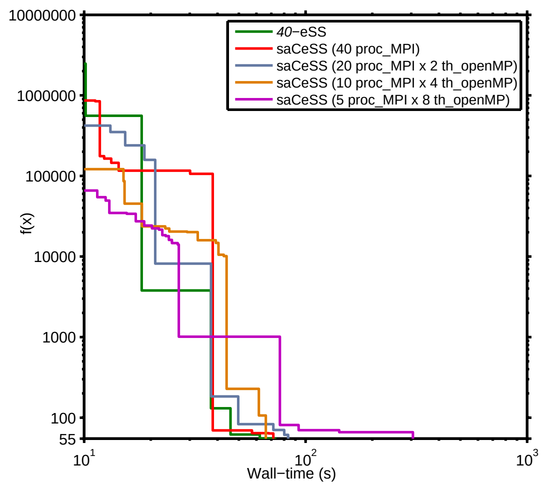
Finally, Figure 3.12 compares the results obtained by the proposed saCeSS method and the results reported in the BioPreDyn-bench suite [220]. Bars show wall-clock times for each solver and benchmark problem, with the numbers over saCeSS bars indicating the overall acceleration obtained. Computations with saCeSS were carried out with 10 processors while computations with eSS were carried out with



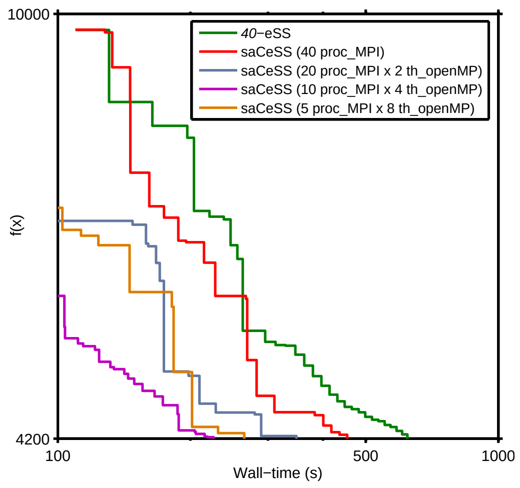
(a) B1



(b) B2



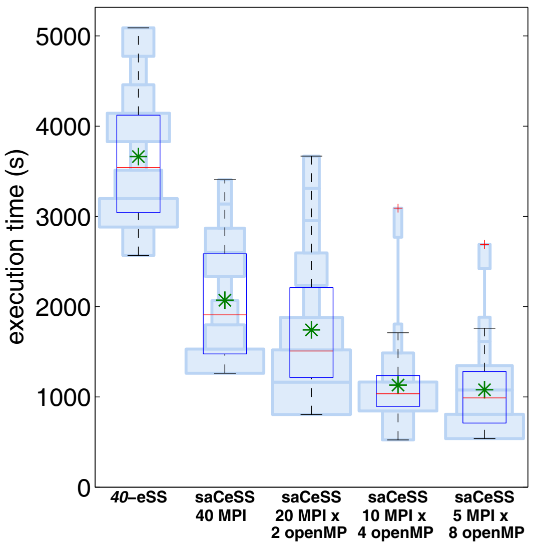
(c) B4



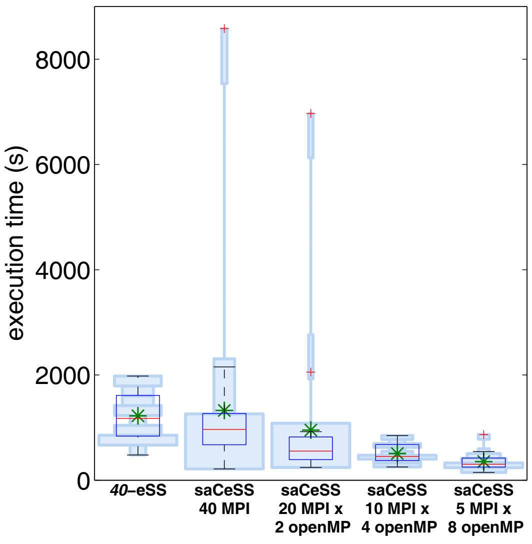
(d) B5

Figure 3.10: Comparative between non-cooperative eSS and saCeSS using 40 processors.

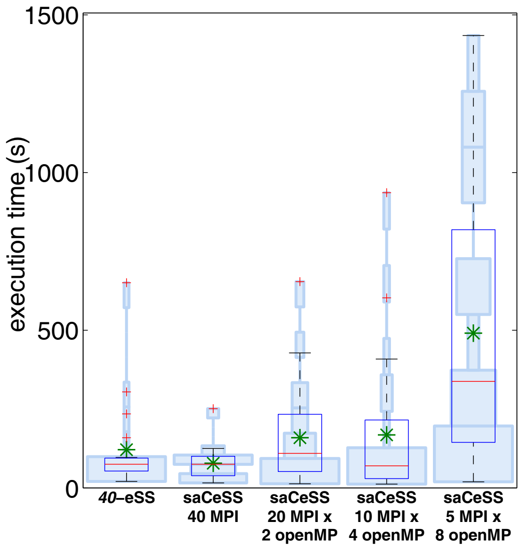
1 processor. Note that, although these figures correspond to a partially different language implementation of eSS (Matlab and C, as reported in [220]), the cost functions and the associated dynamics were computed using the same C code in both cases. It can be observed that the parallel cooperative scheme significantly reduces the execution time of very complex problems as B1 (1 week vs 1 hour) or B3 (10 days vs less than 46 hours). These figures can be especially illustrative for those interested in the potential of HPC in general, and the saCeSS method in particular,



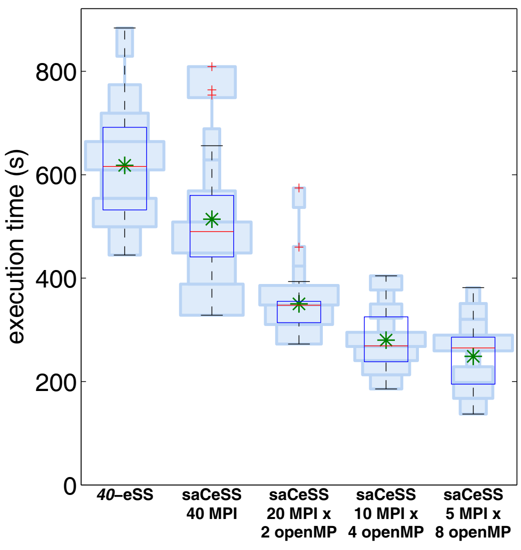
(a) B1



(b) B2



(c) B4



(d) B5

Figure 3.11: Violin/box plots of the execution times of *40*-eSS and saCeSS with different MPI/OpenMP combinations using 40 processes.

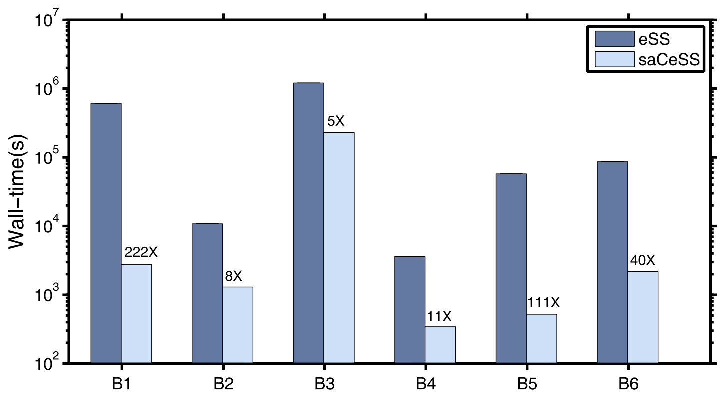


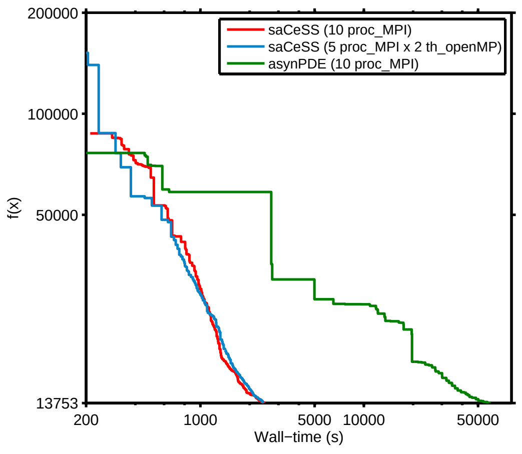
Figure 3.12: Performance acceleration of saCeSS versus eSS (as reported in [220]).

in the solution of parameter estimation problems.

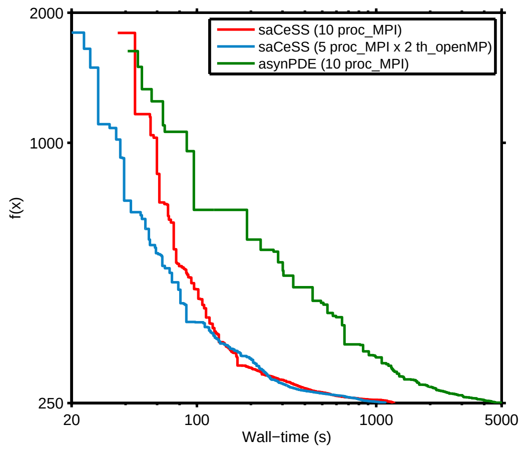
3.4.3. Comparison with other parallel metaheuristics

In order to properly evaluate the novel parallel method presented here, we now compare it with the parallel version of DE proposed in Chapter 2. Previous works in the literature have already pointed out that, for sequential implementations, eSS performs better than DE in those problems where local searches are instrumental in refining the solution [219]. Thus, to ensure a more fair comparison here, we chose our asynPDE algorithm that performs a global search through an asynchronous parallel implementation based on a cooperative island-model, and that also improves the local search phase by means of several heuristics also used in the eSS (i.e. an efficient local solver, a tabu list and a logarithmic space search).

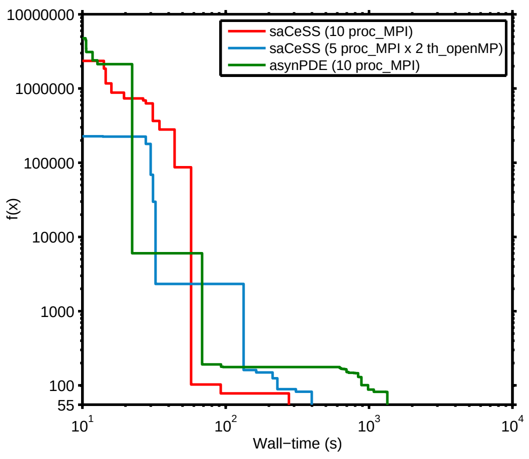
The convergence curves of the asynPDE and the saCeSS algorithms for 10 and 20 processors are shown in Figure 3.13 and Figure 3.14, respectively. These figures represent the convergence curves for those experiments that fall in the median values of the results distribution. Although the best configuration for the saCeSS method is a hybrid MPI+OpenMP one, since the asynPDE method only performs a coarse-grained parallelization, for comparison purposes the convergence curves of saCeSS using only MPI processes are shown. As can be seen, in all cases asynPDE is significantly outperformed by saCeSS.



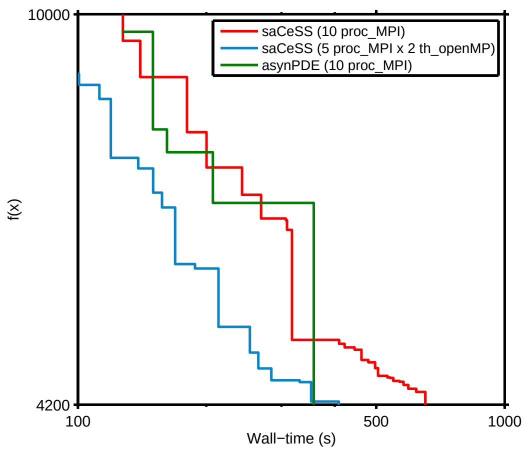
(a) B1



(b) B2



(c) B4

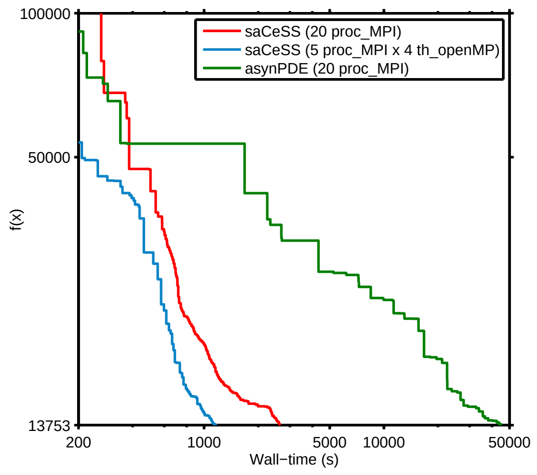


(d) B5

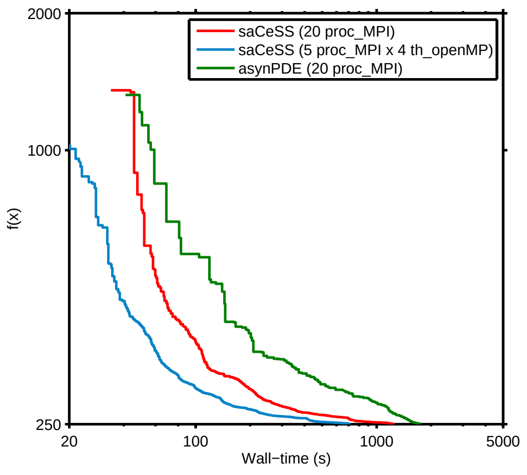
Figure 3.13: Comparison of the convergence curves of asynPDE and saCeSS using 10 processes.

3.5. Concluding remarks

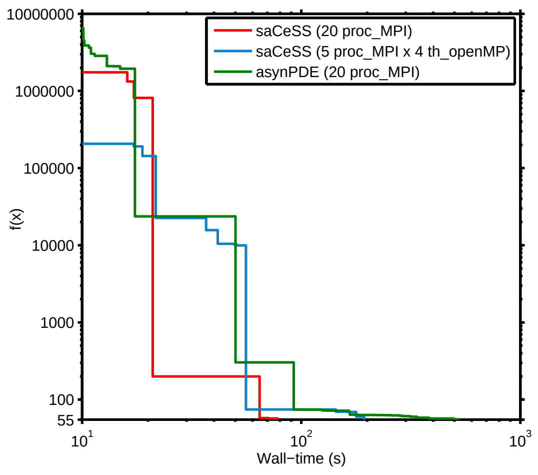
In this chapter we presented a novel parallel method for global optimization in computational systems biology: a self-adaptive parallel Cooperative enhanced Scatter Search (saCeSS) algorithm. A hybrid MPI+OpenMP implementation is proposed combining a coarse-grained parallelization, focused on stimulating the diversification in the search and the cooperation between different processes, with a



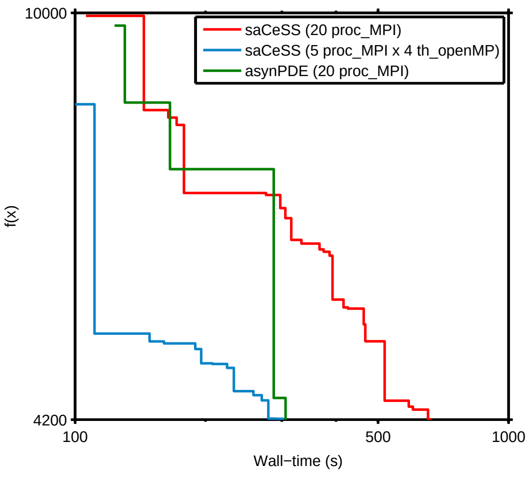
(a) B1



(b) B2



(c) B4



(d) B5

Figure 3.14: Comparison of the convergence curves of asynPDE and saCeSS using 20 processes.

fine-grained parallelization, aimed at accelerating the computation by performing separate evaluations in parallel. This novel approach resulted in a much more effective way of cooperation between different processes running different configurations of the SS algorithm.

The new proposed approach has four main features: (i) a coarse-grained parallelization using a centralized master-slave approach, where the master is in charge of the control of the communications between slaves (islands) and serves as a scoreboard to dynamically tune the settings of the islands based on its individual progress; (ii) a cooperation between processes driven by the quality of the solution obtained during the execution progress, instead of time elapsed; (iii) an asynchronous communication protocol that minimizes processes' halts, thus improving the efficiency in a computational unbalanced scenario; and (iv) a fine-grained parallelization is included in each process to perform separate cost-function evaluations in parallel and, thus, accelerate the global search.

The proposed saCeSS method has been evaluated considering a suite of very challenging benchmark problems from the domain of computational systems biology. The computational results show that saCeSS attains a very significant reduction in the convergence time through cooperation of the parallel islands. A nonparametric statistical analysis shows that the saCeSS method significantly outperforms other parallel eSS approaches, such as an embarrassingly-parallel non-cooperative eSS algorithm, and a previous synchronous cooperative proposal.

The results of this chapter have been published in:

- D. R. Penas, P. González, J. A. Egea, J. R. Banga, and R. Doallo. Parallel metaheuristics in computational biology: an asynchronous cooperative enhanced scatter search method. *Procedia Computer Science*, 51:630 – 639, 2015. Proceedings of the International Conference On Computational Science (ICCS 2015). [166]
- D. R. Penas, P. González, J. A. Egea, R. Doallo, and J. R. Banga. Parameter estimation in large-scale systems biology models: a parallel and self-adaptive cooperative strategy. *BMC Bioinformatics*, 18(1):52, 2017. [167]

The source code of the saCeSS proposed here (see Appendix A) is available at:

- `https://bitbucket.org/DavidPenas/sacess-library`

Chapter 4

Extending saCeSS for mixed-integer non-linear dynamic optimization

During the last decade there has been a growing interest in modelling the dynamics of biological systems. As a consequence, much research effort is now being invested in exploiting these developed dynamic models using mathematical optimization techniques. In Chapter 2 and Chapter 3, we have addressed the solution of non linear programming problems (NLP), i.e. those with continuous decision variables in its entire domain. In this chapter, we will focus on mixed-integer dynamic optimization (MIDO) problems, where part of the decision variables are discrete (binary or integer) [41].

Although many dynamic optimization problems consider how to extract useful operating policies and/or designs from a dynamic model, such formulations can also be applied to the model building process itself, i.e. to the so-called reverse engineering problem [56,75,88,103,113,116,138,188], which is known to be extremely complex [218]. Since the saCeSS method described in Chapter 3 has demonstrated its potential for solving very challenging NLP problems, in this chapter we present a new release, called saCeSS2, derived from modifications and extensions intended to address large mixed-integer nonlinear programming (MINLP) and mixed-integer dynamic optimization (MIDO) problems. In order to illustrate its performance, we consider a set of very challenging reverse engineering problems in the domain of computational systems biology.

The organization of this chapter is as follows. Section 4.1 describes the numerical approach to solve the MIDO problems used in this chapter. Then, Section 4.2 covers the related work. The modifications developed in saCeSS method to work with MIDO-MINLP problems are explained in Section 4.3. Section 4.4 describes the considered applications of the proposed solution within computational systems biology, and Section 4.5 shows the performance and scalability results of saCeSS2 in three challenging case studies. Finally, Section 4.6 summarizes the main conclusions of this chapter.

4.1. Mixed-integer dynamic optimization

The MIDO formulation described in Chapter 1 can be used to solve problems from widely different areas, including aeronautics, chemical engineering, mechanical engineering, transport, medicine, systems biology, synthetic biology and industrial biotechnology [41, 72, 85, 94, 98, 99, 122, 140, 159, 160, 185, 186]. In the particular context of reverse engineering complex biological networks [94], our aim is to use the MIDO formulation to simultaneously identify the underlying network topology, its regulatory structure, the time-dependent controls and the time-invariant model parameters that are consistent with the available experimental (time-series) data. The alternative to this would be to perform real-valued parameter estimation for each possible model structure, which is not tractable in realistic models.

Methods for the numerical solution of dynamic optimization problems can be broadly classified under three categories: dynamic programming, indirect approaches, and direct approaches. Dynamic programming [23, 26] suffers from the so-called *curse of dimensionality*[1], so the latter two are the most promising strategies for realistic problems. Indirect approaches were historically the first developed, and are based on the transformation of the original optimal control problem into a multi-point boundary value problem using Pontryagin's necessary conditions [34,125]. Di-

[1]An exponential increase in volume associated to the search space when the number of dimension is very high, causing sparsity in the distribution of the data [24].

rect methods are based on discretization of the control (sequential strategy [214]), or both the control and the states (simultaneous strategy [28]).

Here we describe the numerical approach used to solve MIDO problems, based on the so-called control vector parameterization direct method. We have chosen the control parameterization approach based on the suggestions given in [41]. This strategy starts by discretizing the control variables (defined as $\mathbf{u}(t)$ and $\mathbf{i}(t)$ in the MIDO formulation described in Chapter 1) into a number of elements and then approximating the controls in each element by means of certain basis functions (e.g. piecewise-constant), in such a way that the control variables are parameterized using $\mathbf{w_u} \in \mathrm{R}^{\rho}$ and $\mathbf{w_i} \in \mathrm{Z}^{\rho}$, which become time-invariant decision variables. With this approach, the original problem is transformed into an outer mixed-integer non-linear programming problem (MINLP) with an inner dynamic system. As a consequence, the evaluation of the objective function and constraints requires the solution of the system dynamics by a suitable initial value problem (IVP) solver.

In summary, the overall strategy followed is composed of:

- An outer mixed-integer nonlinear programming (MINLP) problem. Due to its non-convexity, we need to use global optimization methods. Based on previous experiences with different stochastic global methods and their hybrids with local solvers [21, 60, 61, 67, 165, 167, 189], here we decided to extend the saCeSS method proposed in previous chapter, combining it with an efficient local MINLP solver [69], as described below.

- An inner initial value problem (IVP), i.e. the nonlinear dynamics that need to be integrated for each evaluation of the cost functional and constraints. We solve the IVP using the state-of-the-art solvers for numerical integration of differential equations included in the SUNDIALS package [97]. It should be noted that local optimization methods to be used require the computation of gradients of the objective and/or constraints with respect to the decision variables. If this is the case, an efficient procedure is to use first order parametric sensitivities to compute such information [213]. The sensitivity equations result from a chain rule differentiation applied to the system defined in Eqns. 1.8 with respect to the decision variables and may be solved in combination with the original system. SUNDIALS [97] includes CVODES, an efficient sensitivity solver.

4.2. Related work

Broadly speaking, MIDO problems can be solved using deterministic or stochastic global optimization methods. In the case of deterministic methods, many advances have been made in recent years (see [30, 41, 85, 187] and references therein). Although these deterministic methods can guarantee global optimality in some cases, unfortunately they suffer from lack of scalability, i.e. the associated computational effort increases very rapidly with problem size.

Alternatively, stochastic algorithms for global optimization cannot offer guarantees of global optimality, but usually converge to the vicinity of the global optimum in reasonable computation times, at least for small and medium scale problems. However, for larger problems the computational cost of purely stochastic methods can be very significant [136,141]. Several hybrid approaches [18,21,67,190,192] have tried to benefit from the best of both approaches by combining global stochastic methods with efficient (local) deterministic optimization methods. In this context, metaheuristics (i.e. guided heuristics) have been particularly successful, ensuring the proper solution of these problems by adopting a global stochastic optimization approach, while keeping the computational effort under reasonable values thanks to efficient local optimization solvers [60,181]. Nevertheless, MIDO problems involving a large number of continuous decision variables still tend to have prohibitive solution times, and some convergence difficulties may still be experienced in practice.

As already commented in previous chapters, parallel strategies for metaheuristics have been a very active research area during the last decade. Parallel methods have already shown promising results in NLP problems [108,169]. For the case of MINLP, only a few researchers have considered the development of parallel methodologies. In [83] a parallel implementation of nonlinear branch-and-bound is proposed. The parallelization strategy employs a master-worker paradigm in which the master manages a pool of MINLP tasks corresponding to subtrees of the branch-and-bound tree, and each worker solves a MINLP task by nonlinear branch-and-bound. In [146] a many-core implementation of an adaptive resolution approach to genetic algorithm (arGA) is proposed, to solve both MINLP and non-convex NLP problems. In [157] authors show, using random relaxation procedures, that sampling over a subset of the integer variables and parallel processing have potential to simplify large scale

MINLP problems and, hence, to solve them faster and more reliably than conventional approaches on single processors.

Based on the above, there is still a lack of studies for the MIDO problems considered here. The aim of this chapter is to explore this direction further considering an extension of the saCeSS algorithm proposed in Chapter 3 for NLP problems, so that it can handle general MIDO and MINLP problems of realistic size.

4.3. Extending saCeSS for mixed integer optimization

With the aim of solving the MIDO-MINLP problems presented in Section 4.1, we propose an extension of saCeSS, called saCeSS2, where new functionalities are added to handle this kind of problems. Namely,

- including a local solver for MINLP problems
- changing the self-adaptive mechanism in order to avoid premature stagnation.
- adding new strategies to ensure diversity while keeping parallel cooperation.

Regarding the local search, we have incorporated the *Mixed-Integer Sequential Quadratic Programming* (MISQP) [68, 69] solver. MISQP is a trust region sequential quadratic programming solver adapted to MINLP problems. It assumes that the model functions are smooth: an increment of a binary or an integer variable can produce a small change of function values, though it does not require the mixed-integer function to be convex or relaxable, i.e. the cost function is evaluated only with discrete values in the integer or boolean parameters.

The preliminary tests applying the previous saCeSS algorithm to MIDO-MINLP problems using the MISQP local solver, brought to light a problem of premature convergence due to a quick lose of diversity in the islands. Although both eSS and saCeSS algorithms include their own mechanisms to maintain the desired diversity during the algorithm progress, we observed that in mixed-integer problems a promising incoming solution in an island worked as an attractor for the members of the

RefSet, bringing them quickly to the vicinity of this new value. Thus, we introduced two new strategies in the saCeSS2 method to allow for a dynamic breakout from local optima, and to further preserve the diversity in the search for these problems, avoiding premature stagnation:

- First, we needed to avoid the premature convergence observed in MINLP problems. Cooperation between islands decreases too quickly as the algorithm converges, since many of them stagnate. Thus, the criteria used in the original saCeSS method to trigger the reconfiguration (tuning) of those islands that are not progressing in the search should be accommodated for MINLP problems, relaxing the adaptive conditions to allow for an earlier escape from the stagnated regions.
- Second, we observed that in mixed-integer problems, when an island stagnates, most of the time is due to the loss of diversity in the *RefSet*. Thus, we decided to further inject diversity during the reconfiguration of stagnate islands: once an island requests a reconfiguration, most of the members of the *RefSet*, except for two solutions, are randomly initialized again.

The saCeSS2 scheme follows the same steps as the original saCeSS, explained in Chapter 3. Figure 4.1 shows a schematic, simple representation of the algorithm. For a more in depth view we refer to the pseudocodes in Chapter 3.3. A master process in charge of the control of the cooperation and the scoreboard for the islands' tuning. In the cooperation stage the master manages the appearance of good solutions received from slaves. Then, with the aim of controlling the cooperation between slaves, only when the incoming candidate solution significantly improves a threshold, the solution is updated and broadcasted to the slaves. The master process is able to self-tune the cooperation threshold based on the number of incoming solutions that are refused with the current criterion. In addition, when a new incoming solution deserves to become a cooperative solution spread to the rest of the slaves, there is an increment on the score of the slave that achieved that solution. The master process also manages the slaves' adaptation requests. Each slave decides by itself whether it is evolving in a promising mode or not, and requests from the master the reconfiguration settings.

The slaves perform the classic steps of the sequential eSS. Additionally new

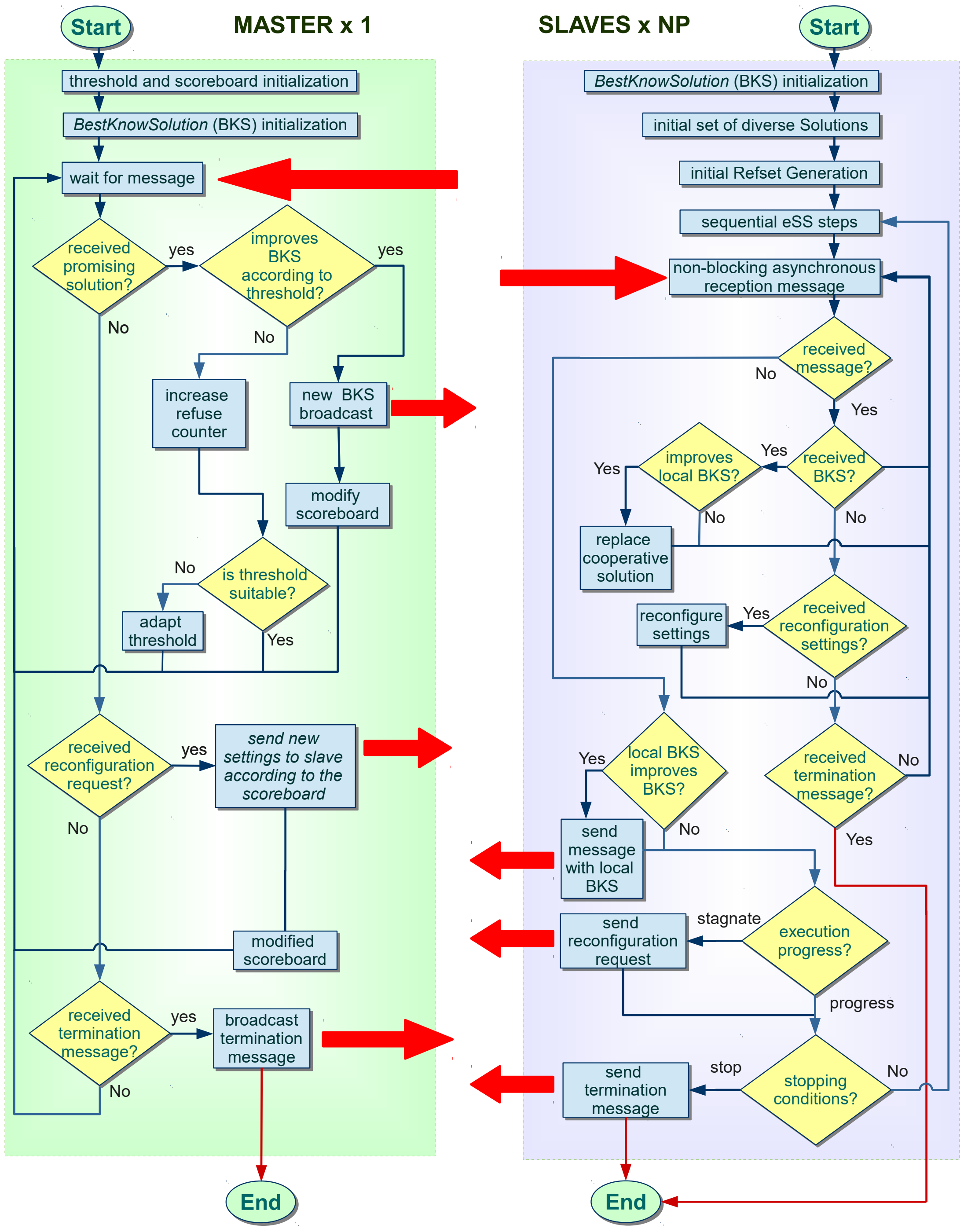

Start
MASTER x 1
SLAVES x NP
Start
threshold and scoreboard initialization
BestKnowSolution (BKS) initialization
wait for message
received promising solution?
yes
No
improves BKS according to threshold?
yes
No
increase refuse counter
new BKS broadcast
modify scoreboard
is threshold suitable?
No
Yes
adapt threshold
received reconfiguration request?
yes
No
send new settings to slave according to the scoreboard
modified scoreboard
received termination message?
yes
No
broadcast termination message
End
BestKnowSolution (BKS) initialization
initial set of diverse Solutions
initial Refset Generation
sequential eSS steps
non-blocking asynchronous reception message
received message?
No
Yes
received BKS?
Yes
No
improves local BKS?
Yes
No
replace cooperative solution
received reconfiguration settings?
Yes
No
reconfigure settings
local BKS improves BKS?
Yes
No
send message with local BKS
received termination message?
No
Yes
execution progress?
stagnate
progress
send reconfiguration request
stopping conditions?
stop
No
send termination message
End

Figure 4.1: Schematic representation of saCeSS2 algorithm.

steps are included to implement cooperation and self-tuning. First, a reception memory buffer retains the messages arriving from the master that have not yet been processed, thus, the communications are all done in a non-blocking asynchronous way. The slave checks if any message with a new best solution from the master has arrived at its reception memory buffer. If a new solution has arrived, the process checks whether or not this new solution improves the local best solution. If the new solution improves the local one, the new solution promotes to be the local best and it replaces the cooperation entry in the process *RefSet*. Then, the slave also checks the reception of new reconfiguration settings. Note that, as already explained, all the communications between slaves and master are asynchronous, thus, the request for a reconfiguration is also a non-blocking operation. This means that the slave goes on with its execution until the message with the reconfiguration settings arrive.

After the reception step, the slave process checks whether its best solution improves in, at least, an ϵ the best known solution. If this is the case, it updates the best known solution and sends it to the master.

Then, the adaptive phase is performed. As previously discussed, this step, specifically the criteria to trigger the reconfiguration in an island, had to be modified to be attuned to MINLP problems. Each slave decides if it is progressing in the search based on:

- Number of evaluations performed since its last cooperation:

  $$N_{eval} > N_{par} \times 500$$

  Where N_{eval} is the number of evaluations performed by this process since its last cooperation with the master and N_{par} is the number of parameters of the problem.

- Balance between the received and sent solutions:

  $$recvSolutions > (4 \times sendSolutions) + 10$$

  Adaptation is requested when the number of received solutions is significantly greater than the number of solutions sent (with a minimum value of 10, to avoid requests at the beginning of the process), that is, if other slaves are cooperating much more than itself.

If an island detects that it is not improving while it is receiving solutions from the master, it sends a request for reconfiguration to the master process. The master listens to these requests and sends to those slaves the most promising settings, i.e. those that are on the top of the scoreboard.

In MINLP problems we further observed a crucial issue: the loss of diversity in the island during the execution progress seriously compromises the convergence. Thus, in order to inject extra diversity into the reconfigured islands, most of the members of their *RefSet* are randomly re-initialized.

Finally, the saCeSS2 algorithm repeats the external loop until the stopping criterion is met. The current version can consider three different stopping criteria (or any combination among them): maximum number of evaluations, maximum execution time and a *value-to-reach (VTR)*. While the VTR is usually known in benchmark problems, for a new problem, the VTR will be, in general, unknown.

4.4. Applications in computational systems biology

The aim of reverse engineering in biological systems is to infer, analyze and understand the functional and regulatory mechanisms that govern their behavior, using the interplay between mathematical modeling with experiments. Most of these models need to explain dynamic behavior, so they are usually composed of different types of differential equations. However, reverse engineering in systems biology has to face many pitfalls and challenges, especially regarding the ill-conditioning and multimodality of these inverse problems [218]. Below we consider several cases related with cell signalling processes and show how these issues can be surmounted with the methodology presented here.

4.4.1. Reverse engineering of cell signalling

Reverse engineering of cell signaling phenomena is a particularly important area in systems biology [7]. In complex organisms, signaling pathways play a critical role in the behavior of individual cells and, ultimately, in the organism as a whole. Cells adapt to the environmental conditions through the integration of signals released by other cells via endocrine or paracrine secretion as well as other environmental stimuli. Fundamental cellular decisions such as replicate, differentiate or die (apoptosis) are largely controlled by these signals [6].

Many of the interactions involved in signaling are commonly grouped in pathways. Pathways are typically depicted as sequences of steps where the information is relayed upon activation by an extracellular receptor promoting several downstream post translational modifications, which will ultimately end by modifying gene expression or some other effector. These interactions are dynamic, i.e. the behavior of such pathways is known to be highly dependent on the cell type and context [107], which change with time [132]. Additionally, many of these pathways interact with each other in ways that are often described as analog to a decision making process [92]. Further, the dynamics of cell signaling are rather fast processes, specially if compared with metabolism or even gene expression.

There are at least three good reasons to infer a dynamic model of a signaling pathway. The first, and perhaps most obvious one, is to find novel interactions. The second is model selection, defined as the process of using data to select (or exclude) a number of model features which are consistent with the current knowledge about a given system. This is particularly relevant when comparing different cell types or a specific cell type in its healthy and diseased status, such as cancer. The third one is the usage of such a model to predict how the system will behave in new conditions that have not been previously tested.

In order to build a mechanistic dynamic model for a given cell type or tissue, we need values for its parameters. These are rarely available, and a common strategy is to find them by training the model to data. The most informative data for signal transduction is obtained upon perturbation experiments, where typically a system (assumed to be homeostatic initially) is stimulated with different chemicals to which the cell may (or not react), and the variations in the cell biochemistry are recorded.

The resulting time-series of data are then used to reverse engineer a dynamic model of the signalling process.

Subsections 4.4.2 and 4.4.3 describe the so called logic-based ordinary differential equations (ODE) framework, which has been found particularly useful in modeling cell signalling, and its problem statement as a MIDO. Then, in Section 4.5 we present three very challenging case studies of increasing complexity, which are then solved with the parallel metaheuristic presented in this study.

4.4.2. Logic-based dynamic models

Logic models were first applied to biological systems by Kauffman [111] to model gene regulatory networks. Since then, applications to multiple contexts have been made [1,223] and diverse modifications from the original formalism have been developed [133]. In particular, various extensions have been developed to accommodate continuous values [8,25,29,51,137,144,232]. Amongst these formalisms, logic-based ordinary differential equations (ODEs) are well suited to handle time series in a precise manner [94]. The main idea is to transform the logic model into a continuous homologue in the form of ODEs. Since it is based on a logic circuit, this formalism does not require mechanistic kinetic information. However, since it is composed of differential equations, we can use it to carry out dynamic simulations and e.g. predict dynamic trajectories of variables of our interest. A number of different methods have been proposed to transform Boolean logic models into ODE homologues [29,137,232].

Basically, logic models describe the flow of information in a biological system using discrete binary states (logic decisions). In other words, each state $x_i \in \{0,1\}$ is represented by a binary variable can be updated according to a Boolean function $B_i(x_{i1}, x_{i2}, ..., x_{iN}) \in \{0,1\}$ of its N inputs (x_{ij}). A typical simple example is the situation where a protein can be phosphorylated in two sites by different kinases, and both interactions are needed to activate the protein. This can be modeled as a logic AND gate. Alternatively, when two different kinases can phosphorylate the same site, independently activating the downstream signaling, we can describe it as a logic OR gate. In another situation, if a signal inhibits the propagation of another one, we can describe it with a NOT gate. In summary, logic models can be

Table 4.1: Relation between functions $B(x_1, x_2)$ and $\bar{B^I}(\bar{x_1}, \bar{x_2})$. A truth table helps to understand the relationship between the OR Boolean update function $B(x_1, x_2)$ and its continuous homologue $\bar{B^I}(\bar{x_1}, \bar{x_2})$. For every combination of the Boolean variables x_1 and x_2, a term is added to $\bar{B^I}(\bar{x_1}, \bar{x_1})$ depending on $B(x_1, x_2)$.

x_1	x_2	$B(x_1, x_2)$	$\bar{B^I}(\bar{x_1}, \bar{x_2}) = ...$
0	0	0	$0\cdot (1 - \bar{x_1})\cdot (1 - \bar{x_2})+$
0	1	1	$1\cdot (1 - \bar{x_1})\cdot \bar{x_2}+$
1	0	1	$1\cdot \bar{x_1}\cdot (1 - \bar{x_2})+$
1	1	1	$1\cdot \bar{x_1}\cdot \bar{x_2}$

represented by an hypergraph with AND/OR/NOT gates.

In the logic-based ODE formalism, we transform each Boolean update function into a continuous equivalent $\bar{B_i} \in [0, 1]$, where the states $\bar{x_i} \in [0, 1]$ can take continuous values between 0 and 1. Their dynamic behaviour is then modelled as:

$$\dot{\bar{x_i}} = \frac{1}{\tau_i}\cdot (\bar{B_i}(\bar{x}_{i1}, \bar{x}_{i2}, ..., \bar{x}_{ij}) - \bar{x}_i) \tag{4.1}$$

where τ_i can be regarded as a sort of life-time of x_i.

HillCubes [232] were developed for the above purpose. They are based on multivariate polynomial interpolation and incorporate Hill kinetics (which are known to provide a good approximation of the dynamics of gene regulation, for example). HillCubes are obtained via a transformation method from the Boolean update function. An example is shown in Table 4.1, illustrating how an OR gate would be transformed by multi-linear interpolation [232] into a BoolCube ($\bar{B^I}$):

$$\begin{aligned} &\bar{B^I}(\bar{x_1}, ..., \bar{x_N}) = \\ &\sum_{x_1=0}^{1} ... \sum_{x_N=0}^{1} \left[B(x_1, ..., x_N)\cdot \prod_{i=1}^{N} \left(x_i\bar{x_i} + [1 - x_i][1 - \bar{x_i}] \right) \right] \end{aligned} \tag{4.2}$$

Although BooleCubes are accurate homologues of Boolean functions, they fail to represent the typical sigmoid shape switch-like behavior often present in molecular interactions [117]. The latter can be achieved by replacing the $\bar{x}_i$ by a Hill function:

$$f^H(\bar{x}_i) = \frac{\bar{x}_i{}^n}{\bar{x}_i^n + k^n} \tag{4.3}$$

or the normalized Hill function:

$$f^{Hn}(\bar{x}_i) = \frac{f^H(\bar{x}_i)}{f^H(1)} \tag{4.4}$$

Further details regarding logic-based ODE models can be found in [232].

4.4.3. Problem statement as a MIDO

In order to find the best logic-based dynamic model to represent the behavior of a given biological network, we can use a formulation extending those in previous works using a Boolean logic framework [183] or a constrained fuzzy-logic formalism [143]. The idea here is that starting from a directed graph containing only the interactions and their signs (activation or inhibition) we can build an expanded hypergraph containing all the possible logic gates.

The problem can be formulated as the following for case studies 1 and 2 (see below):

$$\begin{aligned}
\underset{n,k,\tau,w}{\text{minimize}} \quad & F(n,k,\tau,w) = \sum_{\epsilon=1}^{n_\epsilon} \sum_{o=1}^{n_o^\epsilon} \sum_{s=1}^{n_s^{\epsilon,o}} (\tilde{y}_s^{\epsilon,o} - y_s^{\epsilon,o})^2 \\
\text{subject to} \quad & \mathcal{E}_{sub} = \{e_i | w_i = 1\}, \; i = 1,\dots,n_{\text{hyperedges}} \\
& \mathcal{H}_{sub} = (V, \mathcal{E}_{sub}) \\
& \text{LB}_n \le n \le \text{UB}_n \\
& \text{LB}_k \le k \le \text{UB}_k \\
& \text{LB}_\tau \le \tau \le \text{UB}_\tau \\
& \dot{x} = f(\mathcal{H}_{sub}, \bar{x}, n, k, \tau, t) \\
& \bar{x}(t_0) = \bar{x}_0 \\
& y = g(\mathcal{H}_{sub}, \bar{x}, n, k, \tau, t)
\end{aligned} \tag{4.5}$$

where $\mathcal{H}_{sub}$ is the subgraph containing only the hyperedges ($\mathcal{E}_{sub}$) , defined by the binary variables w. Additionally n, k and τ are the continuous parameters needed for the logic-based ODE approach. These parameters are limited by upper and lower bounds (e.g. LB_k). The model dynamics ($\dot{x}$) are given by the function f. This set of differential equations varies according to the subgraph (and therefore also according to the integer variables vector w). Predictions for the systems dynamics are obtained by solving the initial value problem given by the ODEs. The objective function is the mismatch (e.g. norm-2) between the simulated (y) and the experimental data ($\tilde{y}$), and we seek to minimize this metric for every experiment (ϵ), observed species (o) and sampling point (s). The simulation data y is given by an observation function g of the model dynamics at time t.

In case study 3 we also consider a model reduction problem where additional decision variables are used to remove the influence of a regulator $\bar{x}_i$ from the model. As a starting point we consider a model derived with SELDOM [95], where a mutual information strategy, combined with dynamic optimization, was used to find an ensemble of dynamic models that can explain the data from four breast-cancer cell-lines used in the DREAM-HPN challenge [96]. One of the critical steps in SELDOM was to perform model reduction using a greedy heuristic. Here we consider instead the application of mixed-integer global optimization with saCeSS2 to the problem of model reduction. To find a reduced model we use the Akaike information criterion (AIC), which for the purpose of model comparison is defined as:

$$AIC = 2K + 2n \cdot ln\Big(\frac{F}{n}\Big), \tag{4.6}$$

where K is the number of active parameters. The theoretical foundations for the AIC can be found in [37].

4.5. Experimental results

The new saCeSS2 method described in Section 4.3 has been applied to a set of case studies from the domain of systems biology, as described in Section 4.4, with the goal of assessing its efficacy and efficiency in realistic MIDO-MINLP problems.

The method has been compared with both the sequential eSS [61] and with an embarrassingly parallel non-cooperative version of the eSS called *np*-eSS. The *np*-eSS method consists of *np* independent eSS runs (being *np* the number of available processors) performed in parallel without cooperation among them and reporting the best execution time of the *np* runs. In order to perform a fair comparison, diversity was introduced in these *np* eSS runs in the same sense as cooperative methods do, i.e. each one performing a different eSS with different settings. The performance of saCeSS2 was also evaluated considering a different number of cores in order to study its scalability and the dispersion of results.

Although the reported implementation of eSS [61] was coded in Matlab, to perform here a fair comparison both the eSS and the saCeSS2 algorithms have been implemented in F90. In the saCeSS2 algorithm the MPI library [161] has been employed for the cooperation between islands.

For the experimental testbed different platforms have been used. First, most of the experiments were conducted in the local cluster NEMO that consists of three nodes powered with two deca-core Intel Xeon E5-2650 CPUs with 30GB of RAM connected through a Gigabit Ethernet network. With the aim of assessing the scalability of the proposal we also performed some experiments in a larger infrastructure, the cluster from the European Bioinformatics Institute (EBI) [65], that consists of 222 nodes powered with two octa-core Intel Xeon E5-2680 CPUs with 30GB of RAM, connected through a Gigabit Ethernet network.

The saCeSS2 library has been compiled with the Intel implementations for C, FORTRAN and MPI library, except in the EBI Cluster, where GNU compilers and openMPI had to be used. We remark upon this fact due to the well-known differences in the performance obtained using different compilers.

The computational results shown in this paper were analyzed both from a horizontal view [87], that is, assessing the performance by measuring the time needed to reach a given target value, and from a vertical view [87], that is, evaluating how far a method has advanced in the search for a predefined effort. Thus, two different stopping criteria were considered in these experiments: solution quality based on a *value-to-reach (VTR)*, for an horizontal view, and predefined effort using a maximum execution time, for a vertical approach. The VTR used was the optimal fitness value reported in [94]. Also, since there is a substantial dispersion in the computational results due to the stochastic nature of these methods, each experiment reported in this section has been performed 20 times, with a statistical study also being carried out.

4.5.1. Case Study 1: Synthetic signaling pathway (SSP)

The synthetic signaling pathway (SSP) [133] case study considers a dynamic model composed of 26 ordinary differential equations and 86 continuous parameters. It was initially used to illustrate the capabilities and limitations of different formalisms related with logic-based models. Although this is a synthetic problem, it was derived to be a plausible representation of a signaling transduction pathway. The model was used to generate pseudo-experimental data for 10 combinations of experimental perturbations of 2 ligands (TNFα and EGF) and two kinase inhibitors (for PI3K and RAF1). From a total of 26 dynamic states, 6 were observed (NFKB, P38, AP1, GSK3, RAF1 and ERK) and 5% of Gaussian noise was added to the data.

Following the methodology described in [183], we obtained an expanded version of this model containing every possible AND/OR logic gate given the initial graph structure. This so-called expansion procedure generated a nested model comprising 34 additional variables, one for each hyperedge. Thus, the obtained optimization problem contains 120 parameters, being 86 continuous and 34 binaries. We proceeded by implementing the model and experimental setup using AMIGO [17] and exporting C code which could be used with the saCeSS2 method presented here.

Considering saCeSS2, it is important to note that the cooperation between processes changes the systemic properties of the eSS algorithm and therefore its macroscopic behavior. The same thing occurs with the self-adaptive mechanism proposed. Table 4.2 displays for each method (sequential, parallel non-cooperative, and saCeSS2) the number of cores used (#np), the mean and standard deviation value of the achieved tolerances (*fbest*), the mean and standard deviation number of external iterations (*iter*) performed, the mean and standard deviation number of evaluations (*evals*) required to achieve the VTR, the mean and standard deviation execution time, and the speedup (*sp*) achieved versus the sequential method.

Table 4.2: Case study 1: SSP. Performance analysis from a horizontal view. Stopping criteria: VTR=10.

meth.	**#np**	**fbest±std**	**iter±std**	**evals±std**	**time±std(s)**	**sp**
eSS	1	9.81±0.36	261±636	345989±829560	54885±131153	-
np-eSS	10	9.57±0.63	35±16	356583±127682	4546±1592	12.07
	20	9.81±0.24	29±6	626150±120338	4193±907	13.08
	40	9.86±0.19	33±9	876800±228596	2901±765	18.91
saCeSS2	10	9.77±0.27	25±9	246082±68925	3478±1114	15.77
	20	9.84±0.19	18±6	402613±120260	2779±870	19.74
	40	9.91±0.17	19±8	470746±142702	1602±523	34.25

Table 4.3: Case study 1: SSP. Performance analysis from a vertical perspective. Stopping criteria: VTR=9 and maximum time = 4000 seconds.

meth.	**#np**	**fbest±std**	**iter±std**	**evals±std**	**time(s)**	**hits%**
eSS	1	20.17±4.53	19±3	27289±4478	4000	0%
np-eSS	10	10.68±1.69	26±3	261664±24780	3966	10%
	20	10.31±0.91	24±3	522194±47322	3862	15%
	40	9.57±0.70	36±4	964168±99650	3681	30%
saCeSS2	10	10.28±1.44	23±4	256872±28063	3822	15%
	20	9.61±0.72	22±4	471948±82282	3532	35%
	40	8.95±0.24	32±19	621118±232204	2258	85%

As can be seen, there is a notable reduction in the execution time required by the parallel methods against the sequential one, and there is also a significant reduction between the saCeSS2 method and the non-cooperative *np*-eSS. Note that, in the parallel methods (*np*-eSS and saCeSS2), the initial population, and, thus, the computational load, is not spread among processors. The population size is the same in the sequential method that in each of the islands in the parallel methods. That is, the parallel methods allow for a diversification in the search. Therefore, the speedup achieved versus the sequential method is due to the impact of this diversification, and the speedup achieved by saCeSS2 over the *np*-eSS is due to the impact that the cooperation among processes produces on achieving a good result, performing less evaluations and, hence, providing a better performance. In short, these results show the effectiveness of the cooperative parallel algorithm proposed compared to a non-cooperative parallel version.

Table 4.3 shows results for experiments that include as stopping criterion a prede-

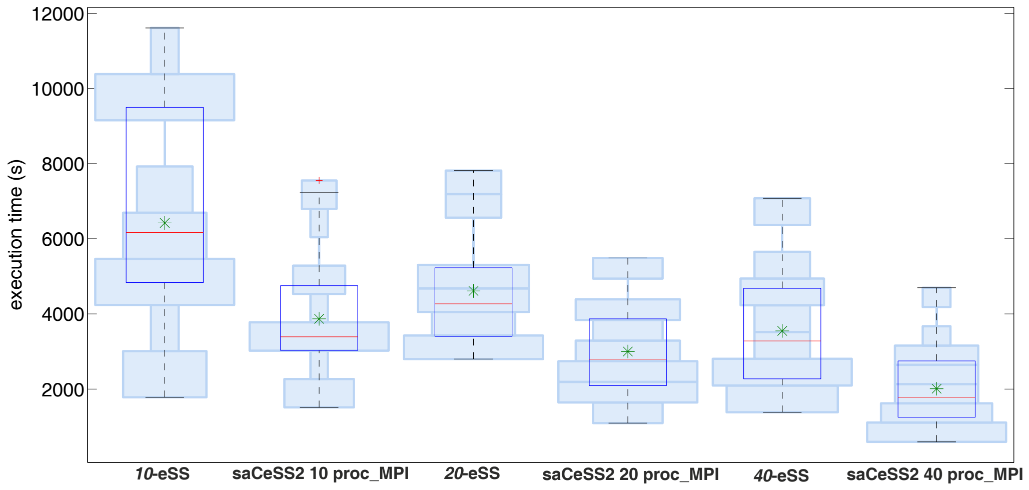
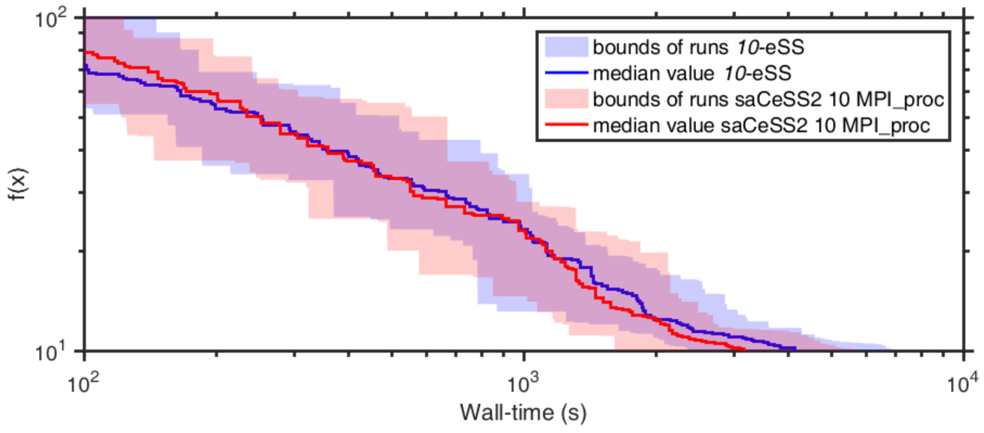


Figure 4.2: Case study 1: SSP. Hybrid violin/boxplots of execution time for *np*-eSS vs saCeSS2 using 10, 20 and 40 MPI processors. Stopping criteria: VTR=10.

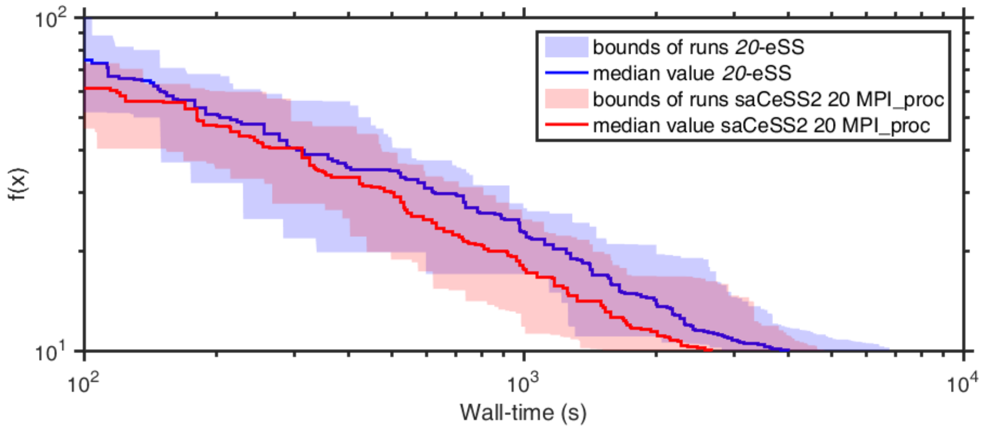
fined effort of maximum execution time of 4000 seconds. This table displays the percentage of executions (% hit) that achieve a very high quality solution (VTR=9.0). It can be observed that the sequential implementation never achieved the VTR in the maximum allowed time, while, for the parallel implementations, when the number of processes grows the number of the executions that achieved the quality solution increased. Again, the cooperative proposed saCeSS2 implementation achieved better results than the non-cooperative parallel version when using the same number of processors.

When dealing with stochastic optimization solvers, it is important to evaluate the dispersion of the computational results. Figure 4.2 illustrates with hybrid violin/boxplots how the parallel algorithms (*np*-eSS and saCeSS2) reduce the variability of execution time and obtain a lower number of outliers when the number of cores increases. The proposed saCeSS2 method outperforms significantly the non-cooperative *np*-eSS method (note the logarithmic scale in axis y). This is an important feature of the saCeSS2, because it reduces the average execution time.

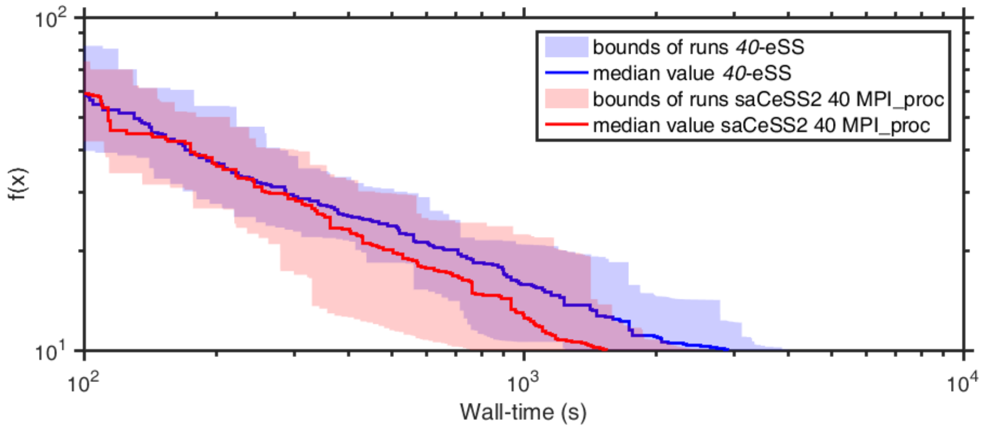
To better illustrate the goal of saCeSS2 method versus the non-cooperative parallel *np*-eSS implementation, Figure 4.3 shows the convergence curves, which represent the logarithm of the objective function value against the execution time. Figure 4.3(a), Figure 4.3(b) and Figure 4.3(c) illustrate, for both saCeSS2 and *np*-eSS



(a) Convergence curves for *10*-eSS vs saCeSS2 using 10 cores.



(b) Convergence curves for *20*-eSS vs saCeSS2 using 20 cores.



(c) Convergence curves for *40*-eSS vs saCeSS2 using 40 cores.

Figure 4.3: Case study 1: SSP. Convergence curves.

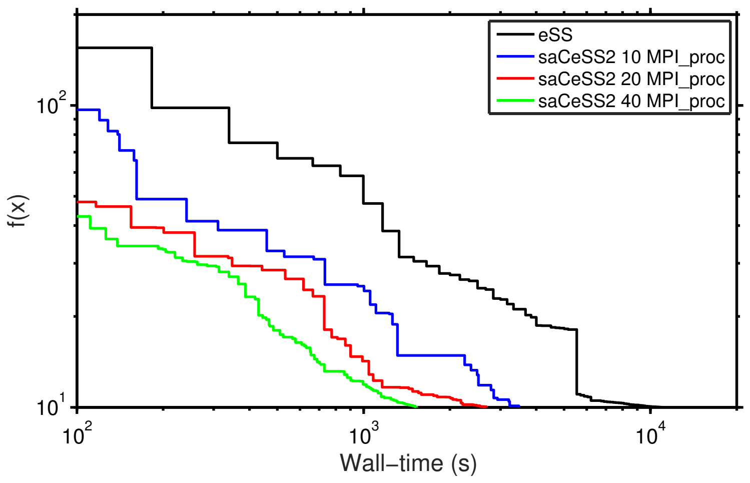


Figure 4.4: Case study 1: SSP. Convergence curves for saCeSS2 using 1, 10, 20 and 40 processors corresponding to the runs in the median values of the results distribution.

methods, the region between the lower and upper bounds of the 20 runs performed for each experiment, with a strong line representing the median value for each time moment.

In order to evaluate the scalability of the proposed saCeSS2, Figure 4.4 shows the convergence curves for those experiments that fall in the median values of the results distribution using 10, 20 and 40 processors. It can be seen that the saCeSS2 still improves the convergence results when the number of processors increases. This improvement comes from the cooperation between islands and the diversification obtained through the exploration in parallel of different search regions using different algorithm settings.

4.5.2. Case Study 2: HePG2

As a second case study, we consider the reverse engineering of a logic-based ODE model using liver cancer data (a subset of the data generated by [10]). The dataset consists of phosphorylation measurements from a hepatocellular carcinoma cell line (HepG2) at 0, 30 and 180 minutes after perturbation.

To preprocess the network, we used CellNOptR, the R version of CellNOpt [202].

Table 4.4: Case study 2: HePG2. Performance analysis from a horizontal view. Stopping criteria: VTR=33.

meth.	**#np**	**fbest±std**	**iter±std**	**evals±std**	**time±std(s)**
eSS	1	-	-	-	-
	10	32.44±0.83	1493±2975	13581782±10598705	230483±365129
np-eSS	20	32.54±0.74	527±381	20267424±12791177	142996±93617
	40	32.89±0.17	434±246	22157687±11660663	70221±35565
	10	32.59±0.59	1056±1873	13637565±20933642	167880±242658
saCeSS2	20	32.38±0.91	396±496	13196677±15577101	89433±108108
	40	32.45±1.03	560±431	11959105±9383238	44037±32346

Basically, the network was compressed to remove as many non-observable/non-controllable species. Subsequently, it was expanded to generate all possible hyperedges (AND gates) formed by a pair of inputs. The obtained full network has a total of 109 hyperedges and 135 continuous parameters. To transform this model into a logic-based ODE model, we developed a parser that generates a C model file and Matlab scripts compatible with the AMIGO toolbox [17].

Consequently, in this case the optimization problem to solve contains a total of 244 parameters, comprised of 135 continuous and 109 binaries. Although the time-series data contains only three sampling time points, it is quite rich from the point of view of information content: it includes 64 perturbations comprising 7 ligands stimulating inflammation and proliferation pathways as well as 7 small-molecule inhibitors blocking the activity of key kinases. To use logic-based ODE models, all data should be in the $[0, 1]$ range and thus we simply normalized the data by rescaling it to this range. From the total of 25 states present in the model, 16 corresponded to observed species. The initial conditions for the other 9 species are not known and were therefore estimated. In order not to increase the problem size and multi-modality unnecessarily, the estimated initials conditions were assumed the same for each of the 64-experiments.

Table 4.4, similarly to Table 4.2, displays the performance of the different methods based on the number of external iterations, function evaluations and total execution time, for a different number of processors. Note that results for the sequential method are not reported due to the unreasonable amount of time to reach convergence. Again, it can be seen that the saCeSS2 method outperforms, not only the

Table 4.5: Case study 2: HePG2. Performance analysis from a vertical view. Stopping criteria: VTR=30 and maximum time = 108000 seconds.

meth.	**#np**	**fbest±std**	**iter±std**	**evals±std**	**time(s)**	**hits%**
eSS	1	48.34±6.27	342±47	1010744±122417	108000	0%
	10	34.91±3.78	482±112	8329751±1519410	103847	10%
np-eSS	20	32.99±1.99	418±65	16612721±2292759	103614	10%
	40	31.12±1.35	604±180	30606532±8576193	93874	35%
	10	34.75±3.88	403±146	7359986±1468640	103052	10%
saCeSS2	20	32.07±4.43	339±163	12057460±4431722	85436	45%
	40	30.41±1.23	786±478	20231060±11664025	63153	65%

sequential eSS, but also a parallel eSS without cooperation between islands. The cooperative strategy, along with the self-adaptive mechanism, leads to an important improvement in the convergence rate and the execution time required.

Table 4.5 shows results using as stopping criterion a lower VTR and a predefined effort of 30 hours. Since it is very difficult to reach a point of very high quality in this problem, this table displays the percentage of hits that achieve a VTR=30. It can be observed that the sequential eSS never achieved the VTR in the maximum allowed time, while the parallel implementations achieve more hits as the number of processors increases. The saCeSS2 method clearly outperforms the embarrassingly parallel eSS.

Figure 4.5 shows violin/boxplots comparing the distribution of the execution times in the saCeSS2 method versus the non-cooperative parallel version. Note the logarithmic scale in y axis. The figure illustrates not only the improvement in the mean execution time, but also the reduction in the variability of the execution times due to the cooperation and self-adaptive mechanism included in the saCeSS2 method. It is worth remarking that the fewer number of cores used the more outliers we obtain in the distribution.

Figure 4.6, demonstrate the scalability of the proposal when the number of processors increases. Finally, Figure 4.7 shows the convergence curves for the previous experiments. Figure 4.7(a), Figure 4.7(b) and Figure 4.7(c) show the region between the lower and upper bounds of the 20 runs for each experiment.

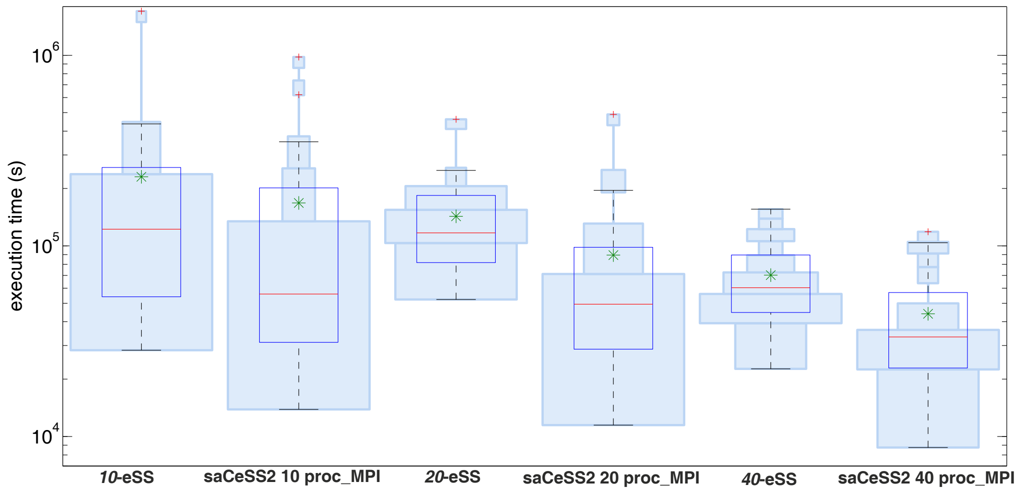


Figure 4.5: Case study 2: HePG2. Hybrid violin/boxplots of execution time for *NP*-eSS vs saCeSS2 using 10, 20 and 40 MPI processors. Stopping criteria: VTR=33.

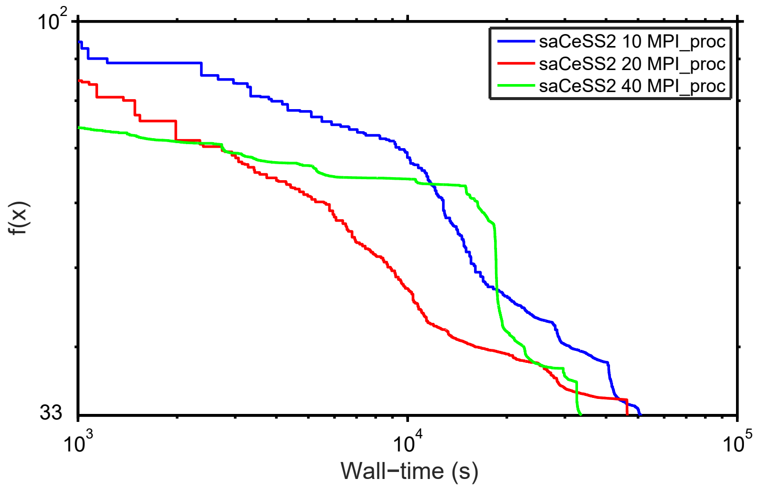
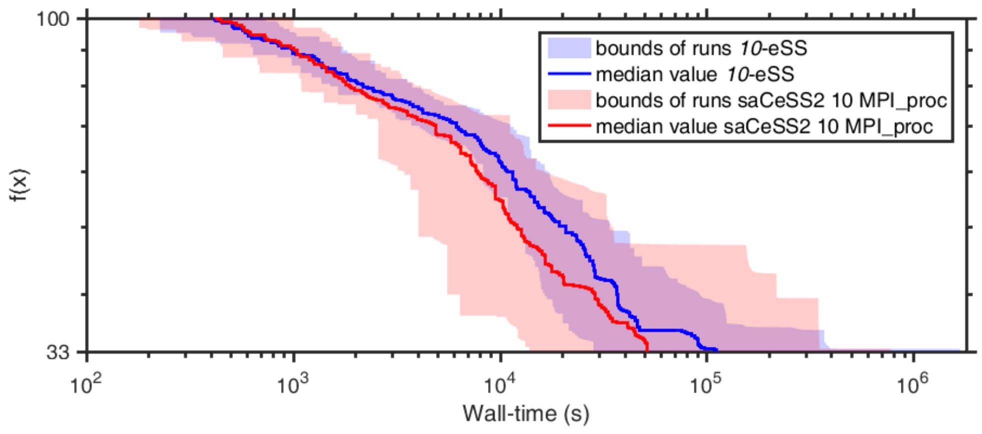
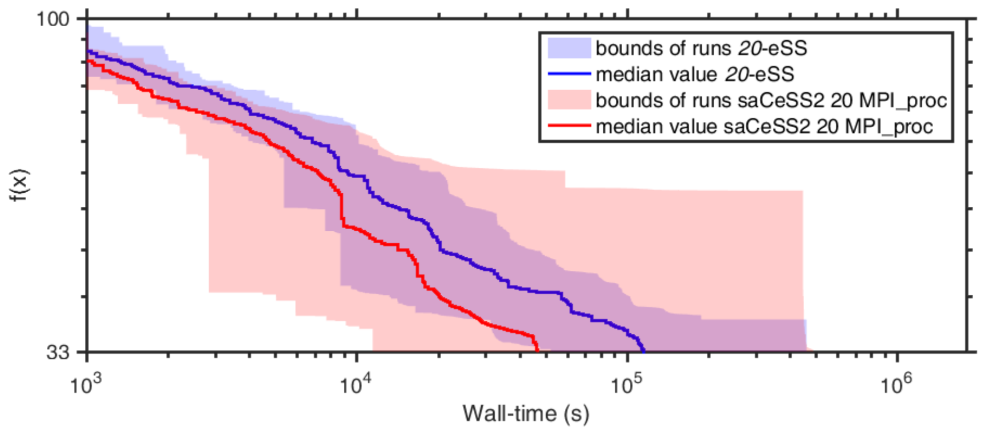


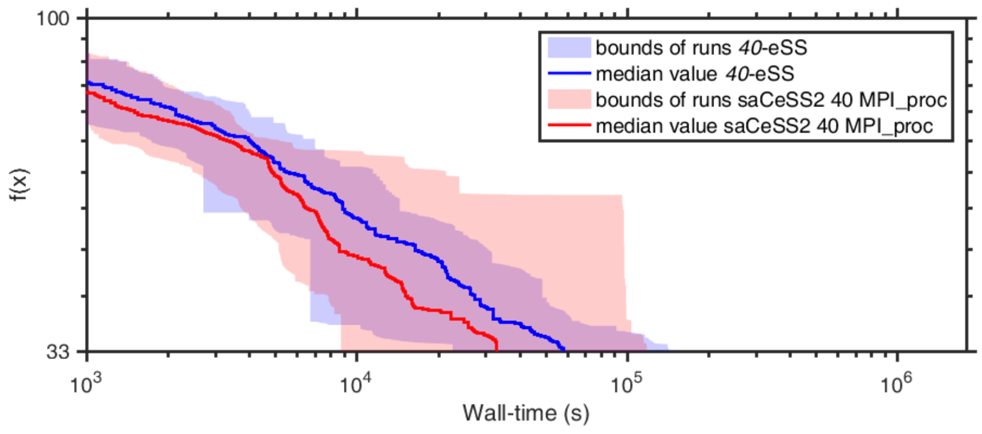
Figure 4.6: Case study 2: HePG2. Convergence curves for saCeSS2 using 10, 20 and 40 processors corresponding to the runs in the median values of the results distribution.



(a) Convergence curves for *10*-eSS vs saCeSS2 using 10 cores.



(b) Convergence curves for *20*-eSS vs saCeSS2 using 20 cores.



(c) Convergence curves for *40*-eSS vs saCeSS2 using 40 cores.

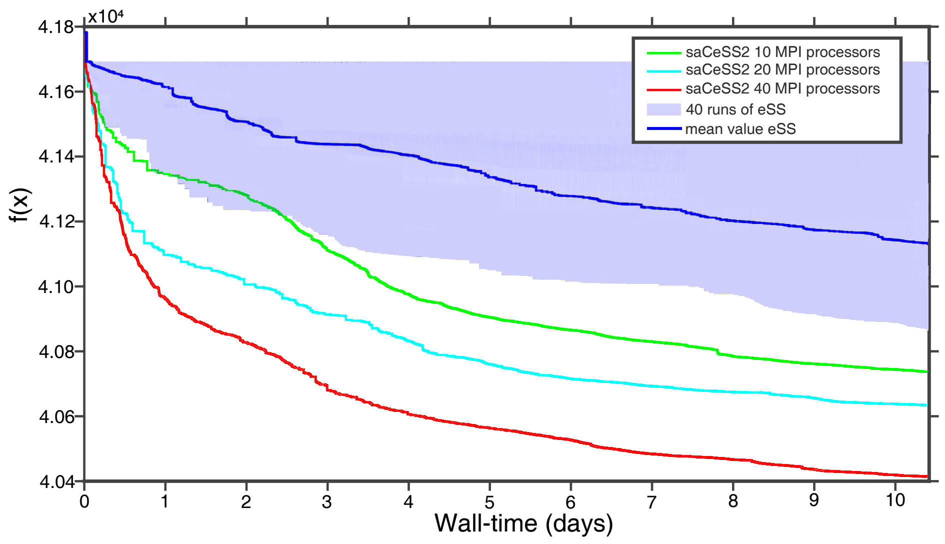
Figure 4.7: Case study 2: HePG2. Convergence curves.

4.5.3. Case Study 3: Breast cancer network inference challenge (HPN-DREAM)

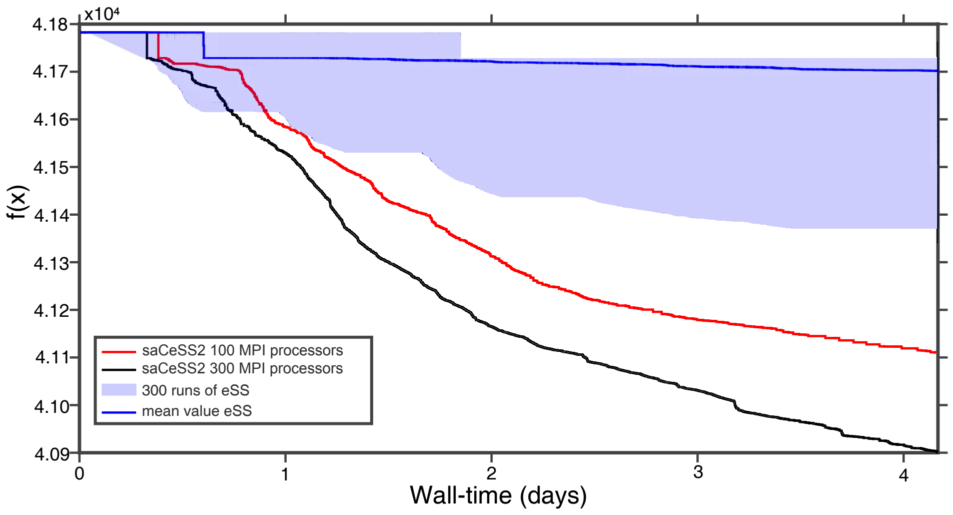
An extremely difficult problem which has been recently made publicly available in the context of the DREAM challenges (www.dreamchallenges.org) is considered in this section. The DREAM challenges provide a forum to crowdsource fundamental problems in systems biology and medicine, such as the inference of signaling networks [96, 174, 184], in the form of collaborative competitions. This data-set comprised time-series acquired under eight extracellular stimuli, under four different kinase inhibitors and a control, in four breast cancer cell lines [96].

The HPN-DREAM breast cancer challenge is actually composed of two sub-challenges: (i) an experimental sub-challenge where the participants were asked to make predictions for 44 observed phosphoproteins (although the complete data-set was larger); and (ii) an *in silico* sub-challenge, where the participants were encouraged to exploit all the prior knowledge they could use and the experimental protocol along with the real names of the measured quantities, used reagents, inhibitors, etc. Using different combinations of inhibitors and ligands (on and off), the organizers of the challenge generated a data-set for several cell-lines. An additional data-set generated with the help of a fourth inhibitor was kept unknown to the participants, who were asked to deliver predictions for several possible inhibitors.

Overall, the problem contains a total of 828 decision variables (690 continuous and 138 binaries). Thus, the HPN-DREAM is an extremely challenging problem also from a computational view, with an enormous expected execution time and an unknown final target value. In a preliminary step, we carried out different experiments using np = 10, 20, and 40 cores in our NEMO local cluster to solve this problem. We used as stopping criterion for all the experiments a predefined effort of 10 days and we studied the convergence curves, shown in Figure 4.8(a). The blue region represents the bounds of the 40 sequential eSS runs, while the blue solid line represents the median value for each time moment of these 40 runs. The other solid lines represent the convergence curve of a single saCeSS2 performed using 10, 20, and 40 cores. The saCeSS2 method clearly outperforms the embarrassingly parallel eSS and shows a good scalability when the number of processes increases. We then performed new experiments using a larger number of cores in the EBI cluster. Figure 4.8(b) show



(a) Convergence curves using 10, 20, 40 and 60 cores in the NEMO local cluster.



(b) Convergence curves using 100, and 300 cores in the EBI cluster.

Figure 4.8: Case study 3: HPN-DREAM. Convergence curves of sequential eSS vs saCeSS2.

the convergence curves using 100 and 300 cores. Due to the large amount of resources employed and the cluster policy, the length of the job (and, thus, the stopping criterion used) had to be set to 4 days. Note that, due to the differences between both infrastructures, it is quite difficult to perform a fair comparison with our local cluster. Although the convergence rate seems to be slower in the EBI cluster, the results obtained still demonstrate the good scalability of saCeSS2. The lower convergence rate in the EBI cluster is due to the architectural and performance differences with respect to our local cluster, and also to the use of GNU compilers instead of the Intel compilers used in our local cluster. Nevertheless, the scalability of saCeSS2 is maintained: the more resources we can use for the cooperative method, the larger improvement we will obtain versus executing the sequential method with the same computational resources.

4.6. Concluding remarks

This contribution extends the previously developed saCeSS method described in Chapter 3, a parallel cooperative strategy for non-linear programming (NLP) problems, so it can successfully solve realistic mixed-integer dynamic optimization (MIDO) problems. To this end, the following features have been included in the new saCeSS2 implementation: (1) an efficient mixed-integer local solver (MISQP), (2) a novel self-adaption mechanism to avoid convergence stagnation, and (3) the injection of extra diversity during the adaptation steps, restarting most of reference set of the reconfigured processes.

The computational results for case studies show that the proposal significantly reduces the execution time needed to obtain a reasonable quality solution. Moreover, the dispersion in the obtained results is narrowed when the number of processors grows. These results confirm that the method can be used to reverse engineer dynamic models of complex biological pathways, and indicates its suitability for other applications based on large-scale mixed-integer optimization, such as metabolic engineering [192], optimal drug scheduling [44, 57] and synthetic biology [158].

The results of this chapter have been sent for publication in:

- D. R. Penas, D. Henriques, P. González, R. Doallo, J. Saez-Rodriguez, and J.

R. Banga. A parallel metaheuristic for large mixed-integer nonlinear dynamic optimization problems, with applications in computational biology. *PLOS ONE*, under review. [168]

The source code of the saCeSS2 proposed here (see Appendix A) is available at:

- `https://doi.org/10.5281/zenodo.290219`

Chapter 5

Evaluation on a Public Cloud Infrastructure

As stated in Chapter 1, Cloud Computing has emerged as a new paradigm for on-demand delivery of computing resources. With the advent of this technology effortless access to large number of distributed resources has become more feasible. However, its adoption by the HPC community has been limited. First, because of the difficulty in employing cloud-based resources. The learning curve to understand the different architectures and runtime environments of various cloud platforms discourage from adopting it as an alternative computational system. Second, because clouds also raise important challenges in performance aspects. Recently, there have been many research works evaluating the promise of cloud platforms for HPC computing, most of them concluding that cloud-based clusters need a significant performance improvement to become competitive for HPC applications.

In this chapter, we evaluate the previously proposed parallel metaheuristics in a public cloud infrastructure: the Microsoft Azure cloud. Additionally, we also present a preliminary comparison of the MPI solution proposed for DE, which is HPC oriented, with another implementation that uses a throughput oriented computing model, Spark. The organization of this chapter is as follows. Section 5.1 briefly describes the related work. Section 5.2 assesses the performance in the cloud of the parallel DE implementation proposed in Chapter 2, including a comparison with a Spark-based implementation. Section 5.3 evaluates in the cloud the saCeSS implementation proposed in Chapter 4 (saCeSS2) for MINLP problems. Finally, Section 5.4 summarizes the main conclusions of this chapter.

5.1. Related work

The evaluation of traditional HPC solutions in cloud environments has received a lot of attention during the last decade. Several researchers have studied the performance of MPI applications in the cloud. Most of these studies use classic MPI benchmarks to compare the performance of MPI on public cloud platforms. The NAS benchmarks have been used in [66], while the Linpack benchmark has been employed in [147]. In [156] a variety of microbenchmarks and kernels were studied. Real applications have also been assessed in the cloud, such as bioinformatics applications [91], high-energy and nuclear physics experiments [112], and different e-Science applications [124, 177]. In [102] the performance of a set of applications that represent the typical workload run at a supercomputing center have been examined. Furthermore, an extensive analysis to detect the more critical issues and bottlenecks of HPC applications in the cloud has been carried out in [70]. All these works conclude that the lack of high-bandwidth, low-latency networks, as well as the virtualization overhead, has a large effect on the performance of MPI applications in the cloud.

Among the new programming models that have been proposed to deal with large scale computations on cloud systems, MapReduce [53] is the most popular one. MapReduce executes in parallel several instances of a pair of user-provided *map* and *reduce* functions over a distributed network of *worker* processes driven by a single *master*. Executions in MapReduce are made in batches, using a distributed filesystem (HDFS) to take the input and store the output. MapReduce has been applied to a wide range of applications, but, when applied to iterative algorithms MapReduce exhibits serious performance bottlenecks [64] mainly because there is no way of reusing data or computation from previous iterations efficiently. New proposals, like Spark [235], aim to provide efficient support for iterative algorithms. According to [235] the performance of iterative algorithms using Spark can be improved by an order of magnitude when compared to MapReduce.

Additionally, in an attempt at converging cloud platforms and HPC, projects like

DataMPI [129] or CloudMPI [3] arose. DataMPI [129] aimed at extending MPI by key-value pair based communication operations to provide high performance communications in cloud scenarios. The cloudMPI [3] framework aimed at designing and implementing an MPI-like framework for cloud platforms, the Azure cloud platform being their preliminary testbed. Unfortunately, none of these projects seems to be active at this moment.

As extensively shown throughout this Thesis, the parallelization of metaheuristics methods has received much attention to reduce the time for solving large-scale problems. However, most of these proposals are parallel implementations based on traditional parallel programming interfaces. Research on cloud-oriented parallel metaheuristics based mainly on MapReduce has also received increasing attention in recent years [105,123,135,176,217]. Some proposals are specific on studying how to apply MapReduce to parallelize the DE algorithm to be used in the cloud. In [239] the fitness evaluation in the DE algorithm is performed in parallel using Hadoop (the well-known open-source MapReduce framework). However, the experimental results reveal that the extra cost of Hadoop DFS I/O operations and the system bookkeeping overhead significantly reduces the benefits of the parallelization. The use of Spark for the parallelization of the DE algorithm was explored in [198]. In this chapter Spark-based implementations of two different parallel schemes of the DE algorithm, the master-slave and the island-based, were proposed and evaluated. Results showed that the island-based solution is by far the best suited to the distributed nature of Spark. Also, a comparison of the previous Spark implementation of the DE algorithm with a MapReduce implementation has been performed in [200], already concluding that Spark outperforms MapReduce in this kind of iterative algorithms.

In this chapter we will explore the implications of the use of MPI and Spark in the parallel implementation of the DE algorithm. We will discuss the differences that arise from the inherent features of each programming model, and we will assess the performance of both implementations in the Microsoft Azure public cloud. Unfortunately, to the best of our knowledge, there are no Spark or Hadoop implementations of SS to compare with the proposed saCeSS method. Nevertheless, an evaluation of the saCeSS method in the Microsoft Azure public cloud has also been done, with the aim of assessing the impact of the overhead related to the virtualization and use of non-dedicated resources in a multitenant platform.

5.2. Parallel DE in the cloud

In this section we will assess the performance of the DE algorithm in the cloud by means of two different implementations: the asynPDE method proposed in Chapter 2, and a Spark-based parallel implementation of the DE algorithm, called eSiPDE, proposed in [201]. First, since the MPI-based asynPDE implementation has already been described in Chapter 2, here we briefly describe the Spark-based implementation remarking the key differences between them. Then, we describe the experiments performed and discuss the results obtained.

5.2.1. Spark island-based Parallel DE

To understand the Spark-based parallel implementation of the DE algorithm (eSiPDE), some previous insights into the way data is distributed and processed by Spark are needed. Spark uses the *resilient distributed dataset* (RDD) abstraction to represent fault-tolerant distributed data. RDDs are immutable sets of records that can optionally be in the form of key-value pairs. Spark programs are run by a driver (the master in Spark terminology) which partitions RDDs and distributes the partitions to workers (the slaves in Spark terminology), that persist and transform them and return results to the driver. There is no communication among workers. Shuffle operations (i.e. join, groupBy) that need data movement among workers through the network are expensive and should be avoided.

With the aim of better understanding Spark intricacies and assess the performance of different alternatives when implementing DE, in [198] a preliminary evaluation of different variants of the master-slave parallel implementation (SmsPDE), and an island-based parallel implementation (this being the previously named SiPDE) have been performed. The main conclusion of that paper is that the island-based parallel implementation is the best suited to the distributed nature of Spark and obtains the best performance results.

The eSiPDE used in the evaluation performed in this chapter follows the scheme shown in Figure 5.1. The algorithm is based on the island model approach, the population matrix is divided in subpopulations where the algorithm is executed isolated. Phases such as selection, recombination and mutation are performed only

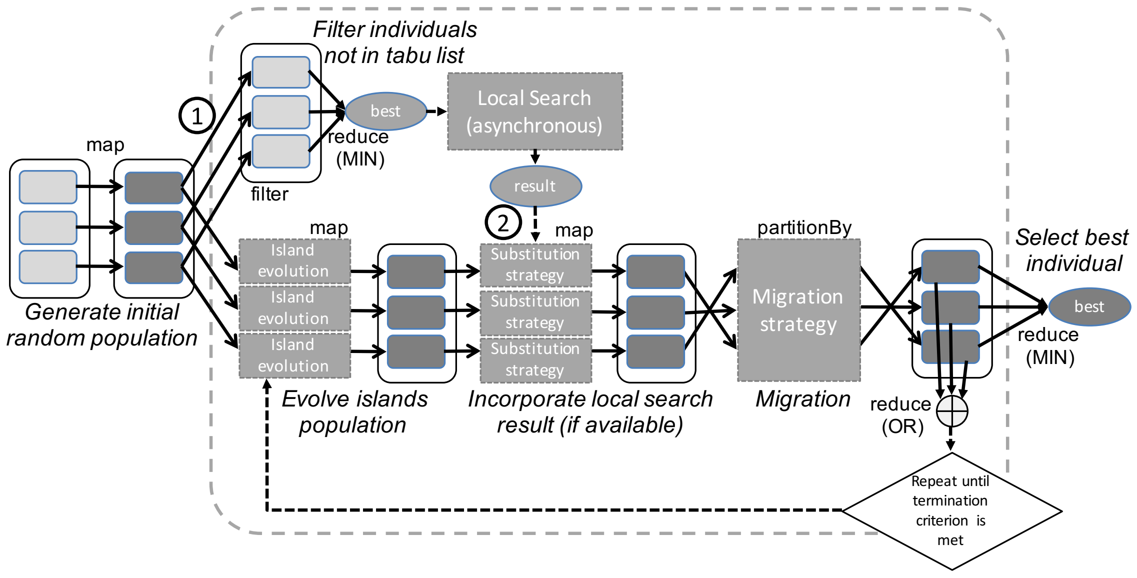


Figure 5.1: Spark implementation of the island-based DE algorithm (eSiPDE).

within each island, which implies absence of collaboration among processes. Sparse individual exchanges are performed among islands to introduce diversity into the subpopulations, preventing search from getting stuck in local optima. In Figure 5.1, boxes with solid outlines are RDDs. Partitions are shaded rectangles, darker if they are persistent in memory. A key-value pair RDD has been used to represent the population where each individual is uniquely identified by its key. There are two execution flows that run asynchronously in different threads of the Spark driver. The main flow is a version of the island-based parallel DE implementation (SiPDE) described in [199], modified to allow for heterogeneous islands and to incorporate to the islands the result of a local search using a substitution strategy. The secondary flow executes an asynchronous local search on the best individual, found up to that moment, that is far enough away from those used in previous searches.

Some steps in the main flow of the algorithm are executed in a distributed fashion:

- The random generation and initial evaluation of individuals that form the population, implemented as a Spark map transformation.
- The evolution of the population. As has been said, the proposed enhanced parallel DE (eSiPDE) is based on the island-based parallel DE (SiPDE) [199],

in which every partition of the population RDD is considered to be an island, all with the same number of individuals. Islands evolve isolated during a number of evolutions. This number can be configured and is the same for all islands. During these evolutions every worker calculates mutations picking random individuals from its local partition only. With this respect, eSiPDE enhances SiPDE by allowing islands to be heterogeneous, that is, having different combinations of CR and F values to enhance diversity.

- The migration strategy, which introduces diversity by exchanging selected individuals among islands every time the evolution of the islands ends. In order to evaluate the communications overhead a custom Spark *partitioner* has been implemented that randomly and evenly shuffles elements among partitions without replacement.
- The checking of the termination criterion, implemented as a Spark reduce action (a distributed OR operation).

The main flow repeats this evolution-migration loop until the termination criterion is met, after which the best individual is selected by means of a Spark reduce action (a distributed MIN operation).

An asynchronous local search runs concurrently with the main flow using a different thread on the Spark driver. As can be seen in Figure 5.1, synchronization with the main flow takes place at two points:

- Before the evolution of the islands (label "1" in the figure), where a new search is initiated if no other is in progress. The candidate solution used as input of the local search would be the best individual, found up to that moment, that was far enough away from candidate solutions used in previous searches. A tabu list is used to keep track of already explored candidate solutions and the input is selected by executing a Spark distributed filtering followed by a reduce action (a distributed MIN operation).
- Once the local search finishes (label "2" in the figure), if it has improved the candidate solution, a substitution strategy is applied in between the evolution and migration steps to incorporate the improved solution into the population.

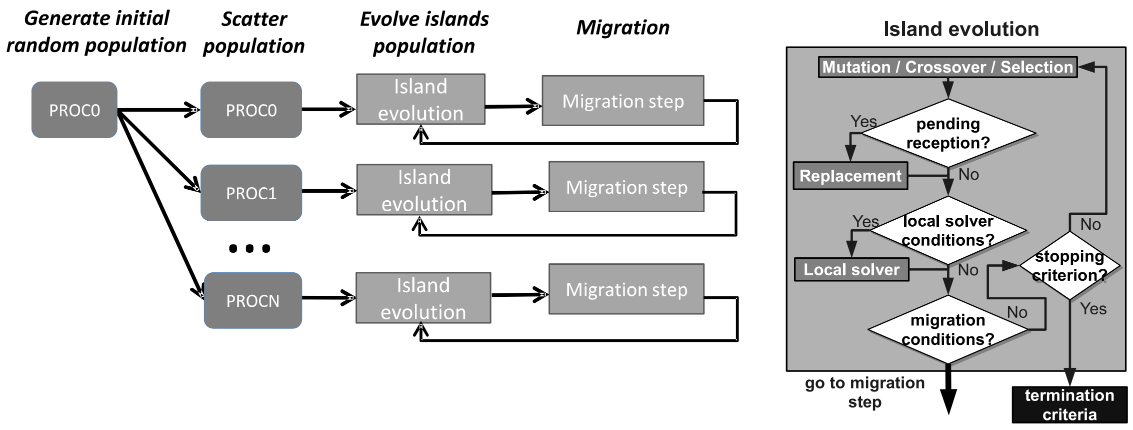


Figure 5.2: Schematic representation of asynPDE.

For this work, a strategy that replaces the worst individual in each island with the local search solution (only if it is better) is used. It has been implemented as a Spark map transformation.

5.2.2. Key differences between the MPI and the Spark implementations

For illustrative purposes, Figure 5.2 shows a conceptual scheme of asynPDE, so as to enable a straightforward comparison with Figure 5.1. There are four main differences between the MPI and the Spark implementations described above. All these differences arise from the inherent features of the programming model used in each implementation, and more specifically from the fact that the communication among workers is not allowed in Spark.

- *Migration strategy.* While in asynPDE the migration strategy consists of a selection of the best individuals in one island to replace the worst individuals in the neighbor, the migration strategy in eSiPDE consists of randomly and evenly shuffling elements among islands without replacement.
- *Synchronization.* The use of a partitioner to perform the migration strategy

leads to a synchronization step in the Spark implementation. The MPI implementation, on the contrary, performs the information exchange between islands through non-blocking asynchronous message-passing operations.

- *Stopping criterion checking.* Although the stopping criterion is evaluated during each island evolution, when it is met by one or more islands the Spark implementation only stops after the reduce operation at the end of the stage. Thus, the Spark implementation cannot stop just right when the stopping criterion is reached as asynPDE code does.

- *Local solver.* While in the asynPDE the local solver is called every m iterations inside each island, in the eSiPDE, there would be at most one local search running concurrently with islands evolution at every moment. If the local search finishes before the islands evolution, its result is incorporated into the population once the evolution ends and a new local search is initiated before the following evolution. On the contrary, if the islands evolution finishes before the local search, a migration is done and a new evolution started without waiting for the local solver to end. This avoids the drawback of synchronous approaches where the evolution of the population gets blocked waiting for a local search to finish. Note also that, in the eSiPDE, the input to the local search is selected from the whole population, and its result is included in every island.

5.2.3. Experimental results

In order to carry out the proposed performance evaluation, the *Circadian* model, already used in the evaluation of Chapter 2, was considered. It consists of a parameter estimation in a nonlinear dynamic model of the circadian clock in the plant *Arabidopsis thaliana*, as presented in [127]. The model contains 7 ordinary differential equations with 27 parameters (13 of them were estimated) with data sets from 2 experiments. This problem is known to be particularly difficult due to its ill-conditioning and non-convexity [141,218]. It must be noted that, as already available implementations in C/C++ and/or FORTRAN existed for this benchmark, we have wrapped it in the Scala code by using Scala native interfaces (i.e. JNI, JNA, SNA). Thus, the code for the benchmark function evaluation has been the same in both Spark and MPI implementations.

For the experimental testbed two different platforms have been used. First, experiments were conducted in cluster Pluton, that consists of 16 nodes powered by two octa-core Intel Xeon E5-2660 @2.20GHz CPUs with 64 GB of RAM, and connected through an InfiniBand FDR network. Second, experiments were deployed with default settings in the Microsoft Azure public cloud using clusters with A3 instances (4 cores, 7GB). For the MPI experiments a custom cluster with canonical Ubuntu Server nodes was used, while for the Spark experiments we used a standard HDInsight Spark cluster with A3 instances for head and worker nodes. All the nodes were located in the North Europe region, although there is no further guarantee on proximity of nodes allocated together, which can lead to significant variability in latency between nodes. Additionally, we had no control over in which underlying hardware the clusters were instantiated on. To avoid ending up with different virtual clusters in every experiment, we instantiated one cluster for MPI experiments and another cluster for the Spark experiments and all the tests have been performed in these clusters. By examining /proc/cpuinfo we have identified that the actual CPU used in both clusters was an Intel Xeon E5-2673 @2.40GHz with 7GB of RAM.

As mentioned in Chapter 2, there are many configurable parameters in the classic DE algorithm, such as the population size (NP), the mutation scaling factor (F), the crossover constant (CR) or the mutation strategy (MSt), whose selection may have a great impact in the algorithm performance. The objective of this work is not to evaluate the impact of these parameters, thus, only results for one configuration are reported here. For the selection of the settings in these experiments, the guideline in [194] has been followed. For all the experiments in this section NP=256, F=0.9, CR=0.8, and MSt=DE/rand/1 were used. The target value, or value-to-reach (VTR), used as stopping criterion in the following experiments was 1.0e-5.

As well, in parallel island DE algorithms, new parameters have also to be considered, such as the migration frequency (μ), the island size (λ), the communication topology between processes, or the selection and replacement policy in the migration step. The migration frequency will have a significant impact in the performance of both implementations, because a high migration rate will emphasize the communications overhead, particularly affecting the performance on cloud platforms. Thus, we performed a preliminary study to determine the optimal migration frequency for

Table 5.1: Performance evaluation of asynPDE and eSiPDE in local cluster Pluton.

meth.	#cores	#evals	time±std(s)	speedup
asynPDE	1	6480102	15230.22±886.80	-
	2	3540889	4078.36±1852.32	3.73
	4	1815689	1100.08±180.96	13.84
	8	1231094	380.99±77.64	39.97
	16	1236346	220.79±51.17	68.98
	32	1700782	149.82±30.37	101.65
eSiPDE	1	6437670	40883.39±3712.56	-
	2	5980416	19275.65±1281.63	2.12
	4	5729536	9305.30±909.41	4.39
	8	3904256	3319.33±296.88	12.32
	16	1835776	790.97±90.50	51.69
	32	1577216	348.36±43.47	117.36

both implementations. A migration frequency of 50 iterations between migrations for asynPDE, and of 200 iterations between migrations for eSiPDE, has been chosen. In addition, the island size will be $\lambda = NP/\#cores$. Finally, in the MPI implementation, the communication topology used is a star, and the selection policy consists in selecting only the best individual in the island population to be sent as a promising solution, while the replacement policy consists in replacing the worst individual in the island population with the incoming solution. Note that in the Spark implementation the migration step consists in a shuffle of the island populations instead of a selective send and replacement in each island.

Comparing the different implementations of the parallel metaheuristic is not an easy task due to their key differences that affect the convergence rate of the algorithms. So we carried out two different experiments. First, we have analysed the performance of both approaches without the local solver and tabu list features. Thus, a more accurate discussion can be made regarding the differences introduced by the cooperation scheme in both implementations. Second, we activated the local solver and tabu list features, so we can discuss the differences introduced by the strategy that the local solver follows in each implementation.

Results in cluster Pluton for both asynPDE and eSiPDE implementations, with the local and tabu list disabled, are shown in Table 5.1. We carried out experiments varying the number of cores, from 2 to 32. We have not used more than 32

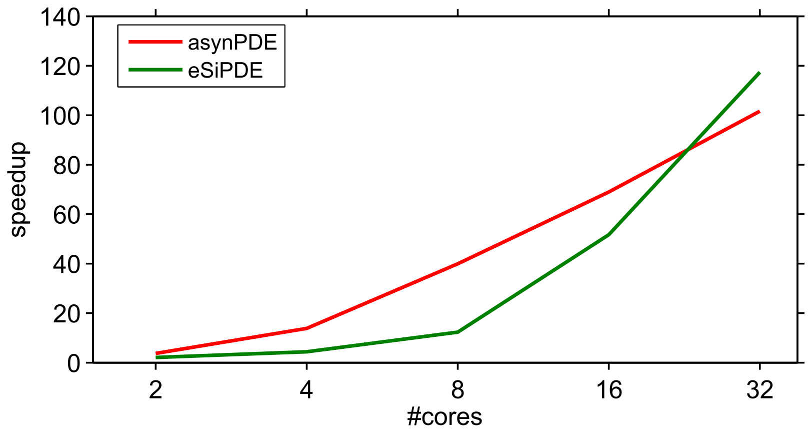


Figure 5.3: Speedup achieved by asynPDE vs eSiPDE in Pluton.

cores because the scalability of the parallel DE algorithm is heavily restricted by the population size, and, according to the guideline of [194], the population size should be around 10D (being D the dimension of the problem, which in the case of the Circadian benchmark is 13). We have set the population size to 256 individuals, so as to be able to scale up to 32 islands of $\lambda = 8$ individuals each. And, although some authors [149] have shown that a population size lower than the dimensionality of the problem can be optimal in many cases, the fact is that the smaller the island population size is, the less chances for the combination between individuals and a lower convergence rate will be achieved. Table 5.1 displays, for each experiment, the number of cores (*#cores*) used, the mean number of evaluations required (*#evals*), the mean and deviation of the execution times (*time(s)*), and the speedup achieved versus the sequential execution. As we already knew from Chapter 2, results show that the parallelization improves the execution time required for convergence, not only by performing the evaluations in parallel but also because the cooperation between islands leads to an improvement in the convergence rate (fewer evaluations are needed), thus, achieving superlinear speedups. As can be seen, the convergence rate in both implementations differs when the number of islands increases. Although for a small number of islands the MPI implementation clearly outperforms in number of evaluations to the eSiPDE implementation, it should be noted that the convergence of the asynPDE implementation stagnates for more than 8 processes, while it improves for the Spark implementation. This can be also observed in the speedup trend, shown graphically in Figure 5.3. This important feature is due to differences in the migration strategy followed by both implementations. When the number of islands increases, selecting the best individuals to be shared in the migration step leads to small populations full of cooperative solutions that has an adverse impact on diversity, and could even cause premature convergence to local optima. Shuffling the islands population, on the contrary, maintains the diversity of the searches when the number of islands increases.

The same experiments were performed in the Azure public cloud from 2 to 16 cores. Results are shown in Table 5.2. As can be seen, the Azure experiments obtain similar results as those carried out in the local cluster in terms of convergence (number of evaluations required). However, results in terms of execution times and speedup differ. On the one hand, the overhead introduced in Azure due to virtualization and use of non-dedicated resources in a multitenant platform are not negligible, the execution times obtained in Azure being between 2x and 3x those obtained in Pluton. On the other hand, the speedups achieved in Azure, in particular when the number of cores grows, are larger than in Pluton. This is due to the computation-to-communication ratio, that is, the ratio of the time spent computing to the time spent communicating, which depends on the relative speeds of the processor and the communication medium. In particular, for the asynPDE implementation the number of communications increases with the number of cores, and the computation, on its turn, decreases. Thus, since the computation is slower in Azure, the scalability is better in this platform.

To further compare the performance of both implementations without attending

Table 5.2: Performance evaluation of asynPDE and eSiPDE in Azure public cloud.

meth.	**#cores**	**#evals**	**time±std(s)**	**speedup**
asynPDE	1	6633830	37952.61±3224.67	-
	2	3067622	9196.63±1110.82	4.13
	4	1809942	2659.65±410.31	14.27
	8	1279609	929.77±204.21	40.82
	16	1301888	491.92±87.50	77.15
eSiPDE	1	6565461	93977.02±5216.28	-
	2	5333186	41140.87±6474.26	2.28
	4	5716736	21030.04±2443.06	4.47
	8	3983616	7444.79±928.91	12.62
	16	1953536	1768.25±166.51	53.15

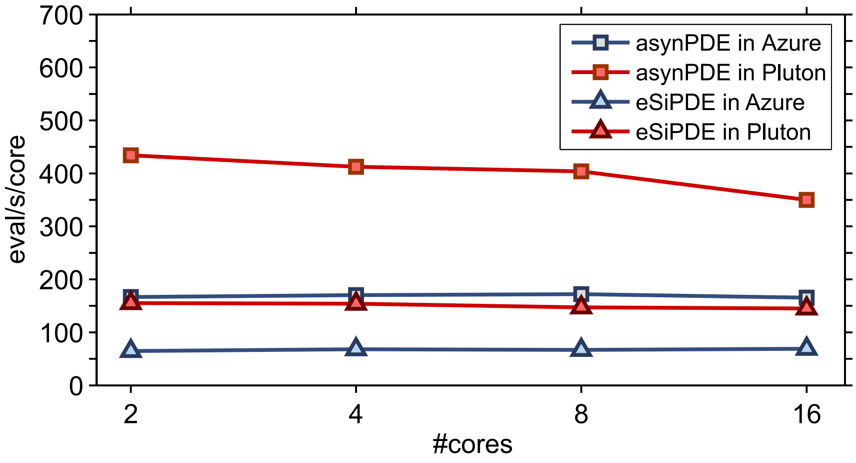


Figure 5.4: Eval/s/core achieved by asynPDE vs eSiPDE in local cluster Pluton and Microsoft Azure public cloud.

to the convergence rate achieved in each case, the number of evaluations per second and per core (eval/s/core) has been calculated. Note that this computation includes not only the CPU time for the evaluation itself but also the communication time and other overhead introduced by the algorithm implementation, thus, it is a good metric to assess the performance of Spark versus MPI for this problem. Figure 5.4 shows the eval/s/core achieved for both implementations and the two infrastructures used. We encountered that the eval/s/core of the MPI implementation was between 2.1x and 2.5x the one obtained by the Spark implementation. A drop can also be observed in the number of evaluations per second and core in Pluton for the asynPDE implementation when the number of cores grows. The reason is that the number of communications, and thus their overhead, increases with the number of cores in the MPI implementation, and, additionally, since the computation of each island decreases with the number of cores, the trade-off between computations and communications degrades. However, in the Spark implementation the number of communications in each shuffle is always the same, and this data movement is spread among the number of cores, thus, providing a good scalability.

After assessing the performance of the cooperation scheme in both implementations, we performed the experiments enabling the local solver and tabu list enhancements. Results for these experiments are reported in Table 5.3. Table 5.3 displays, for each experiment, the number of cores (*#cores*) used, the mean number

Table 5.3: Comparison of asynPDE versus eSiPDE with local solver and tabu list enabled.

	method	**#cores**	**#evals**	**time±std(s)**
Pluton	asynPDE	2	78276	94.62±66.75
		4	78903	49.23±35.79
		8	79992	26.38±21.12
		16	87341	17.13±14.11
	eSiPDE	2	179456	472.41±441.29
		4	230656	388.31±736.39
		8	171776	134.01±140.78
		16	225536	115.48±119.04
Azure	asynPDE	2	70332	201.49±110.32
		4	57195	84.68±2.55
		8	69469	54.45±22.06
		16	72244	30.54±5.07
	eSiPDE	2	102656	745.88±656.98
		4	120576	453.78±431.47
		8	156416	355.51±347.84
		16	135936	160.30±151.85

of evaluations needed (*#evals*) used, and the mean of the execution times (*time(s)*). As can be observed in both implementations, the reduction in the number of required evaluations is significant. However, this improvement is larger in the MPI implementation. It converges between 5 and 7 times more quickly than the Spark implementation, mostly because it achieves a reduction in the number of function evaluations required between 2x and 3x. Two are the main causes. First, because although the stopping criterion is evaluated during each island evolution, when it is met by one or more islands the Spark implementation only stops after the reduce operations at the end of the stage (see Figure 5.1). Thus, if the stopping criterion is met by the local solver, which in the Circadian problem occurs frequently, the Spark implementation cannot stop until the end of the islands evolution. Second, because in the Spark-based implementation only one local solver is running at each time, while in the MPI implementation each island executes its local solver, thus, augmenting the probabilities of finding the target value. These results can be useful to guide the improvement of the Spark implementation by means of increasing the number of local solvers that can run concurrently.

5.2.3.1. Performance/cost evaluation

Before finishing the evaluation we wondered how much performance we can obtain when using larger (and more expensive) instances in the Azure platform. Azure provides a number of different instance types that have varying performance characteristics and prices. Previous results were obtained in clusters of A3 instances. For this new comparison, we have also built a cluster with compute-intensive instances where we have carried out the tests with the MPI implementation. We have chosen A11 compute-intensive instances that are 16-core nodes with Intel Xeon E5-2670 @2.6GHz CPUs with 112GB of RAM. Figure 5.5 shows the results obtained and its comparison with the results obtained in the previous Azure A3-cluster and also in the local cluster Pluton. For the fairest comparison, the local solver and the tabu list were disabled. It can be seen that execution times (shown in a logarithmic scale in the primary axis) in the A11-cluster are competitive with those obtained in the local cluster, and even outperforms Pluton when the number of cores grows, showing a better scalability. The number of evaluations per second and core is also shown in the secondary axis and clearly illustrate the improvement in scalability when using the compute-intensive Azure instances. If we take a look to the price of these instances, we can see that in November 2016 the cost of each A3-instance was 150.58

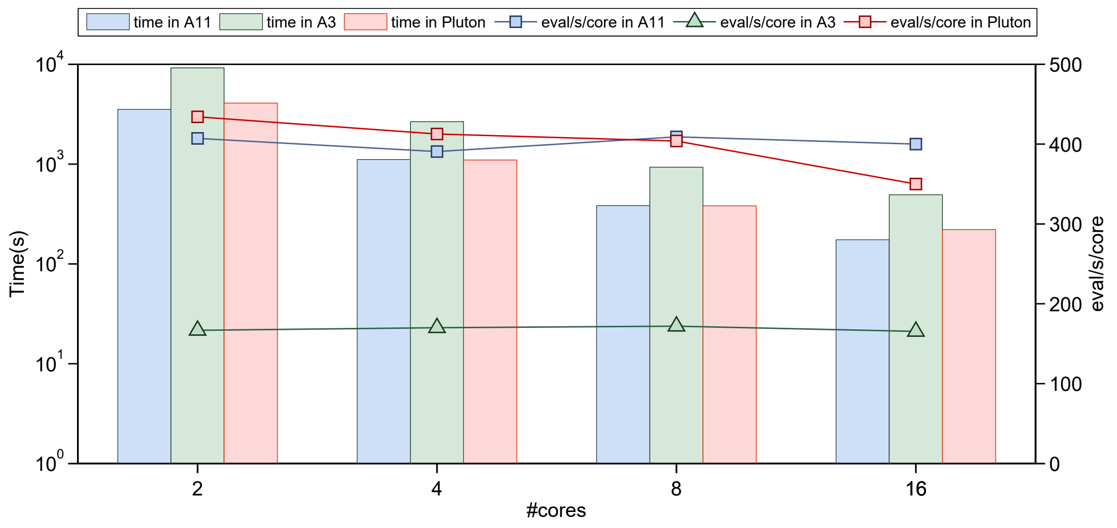


Figure 5.5: Comparison of asynPDE results in local cluster Pluton with results in a cluster of A3 instances and a cluster of A11 instances in Azure.

EUR/node/month, while the cost of the A11-instance was 978.77 EUR/node/month. To run the experiments in this section we used 16-core clusters, so we need four A3 instances which results in an estimated cost of 600 EUR/month, and we need only one A11 instance which has an estimated cost of 980 EUR/month. Taking into account that the performance, in terms of execution time, of the A11-cluster in the tests performed in this work has been 2.5x over the A3-cluster, we can conclude that the savings using A11 instances over A3 in this case-study would be around 35%.

5.3. Evaluation of saCeSS method in the cloud

As was already demonstrated in Chapter 3 and Chapter 4, though saCeSS clearly outperforms the sequential and the non-cooperative parallel versions of the eSS, it still requires large computational times to achieve convergence in very complex problems. Additionally, it has been shown that the diversity introduced by the increase in the number of islands clearly improves the algorithm convergence rate. However, an increase in the number of islands should be attended of an increase in the number of computational resources (cores), and this is not always practicable.

The cloud may help in solving this issue by providing effortless access to a larger number of distributed resources. For this reason, we decided to evaluate the applicability and performance of saCeSS2 method in a cloud platform, comparing the results obtained with those discussed in Chapter 4 achieved in a local cluster.

5.3.1. Experimental results

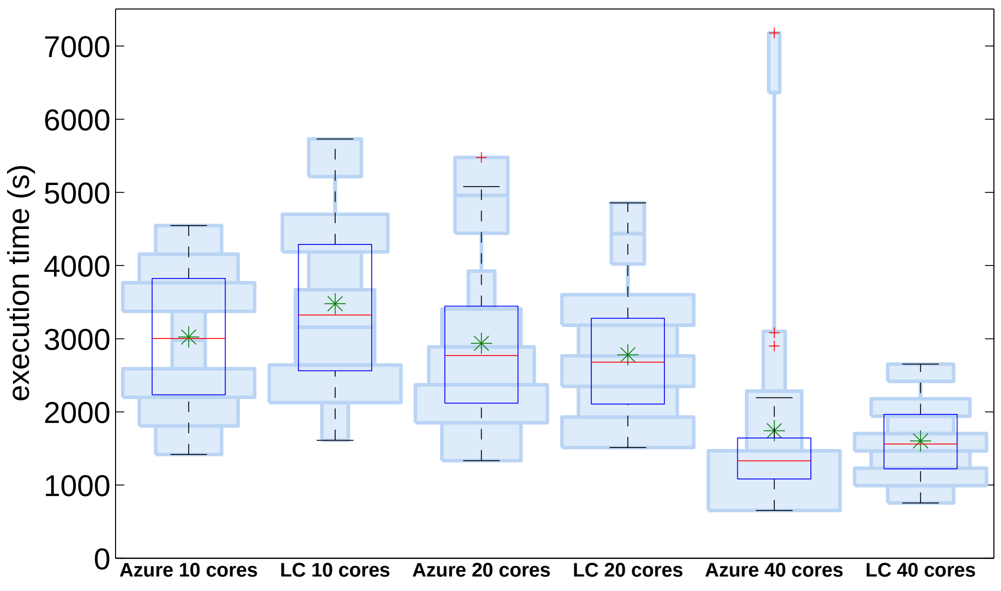
Experiments were deployed in the Microsoft Azure public cloud using clusters with compute-intensive A9 instances (16 cores, 112GB). These instances are designed and optimized for compute-intensive and network-intensive applications. Each A9 instance uses an Intel Xeon E5-2670 @2.6GHz CPUs with 112GB of RAM. Additionally, A9 instances feature a second network interface for remote direct memory access (RDMA) connectivity. This interface allows instances to communicate with each other over an InfiniBand network, operating at QDR rates, boosting the scalability and performance of many MPI applications.

Table 5.4: Performance of saCeSS for SSP and HePG2 case studies. Stopping criteria: $VTR_{SSP} = 10$ and $VTR_{HePG2} = 33$.

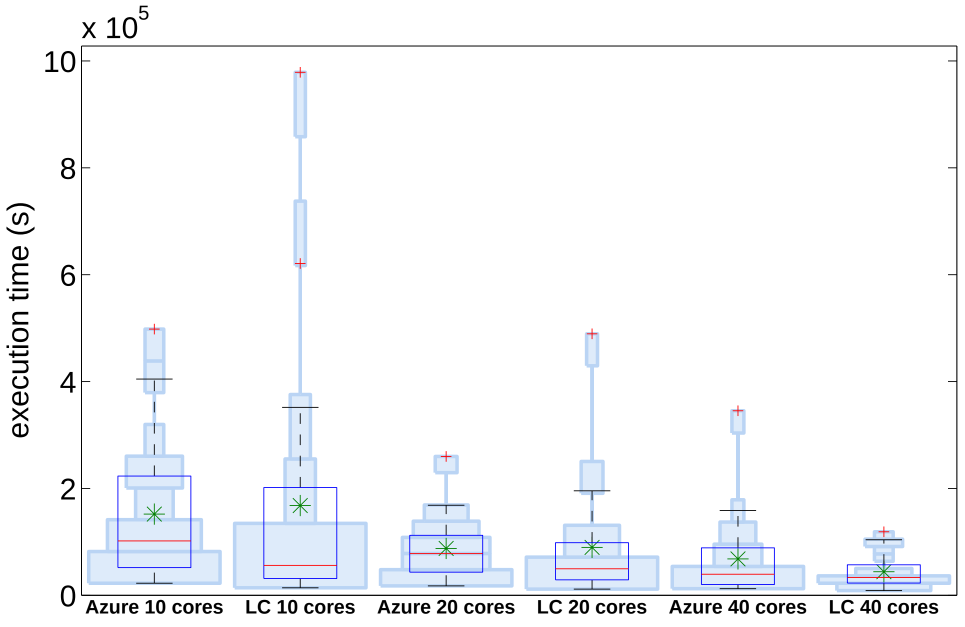
problem	**infr.**	**#cores**	**iter±std**	**evals±std**	**time±std(s)**
SSP	Azure	10	23±8	246256±73545	3153±948
		20	21±9	470857±175723	3057±1177
		40	31±44	571966±423381	1861±1426
	cluster	10	25±9	246082±68925	3478±1114
		20	18±6	402613±120260	2779±870
		40	19±8	470746±142702	1602±523
HePG2	Azure	10	807±782	11790096±10574631	151939±128256
		20	305±243	10617112±7403497	87802±58619
		40	731±707	17438937±19853336	68214±77442
	cluster	10	1056±1873	13637565±20933642	167880±242658
		20	396±496	13196677±15577101	89433±108108
		40	560±431	11959105±9383238	44037±32346

Benchmarks SSP and HePG2, already used and described in Chapter 4, have been evaluated. Table 5.4 shows the performance of the saCeSS method for both case studies in the Azure public cloud. In order to ease the comparison, we include in this table the results obtained in the local cluster, that have already been reported in Tables 4.2 and 4.4 of Chapter 4. As can be seen, the behavior of the algorithm differs slightly from the results obtained in the local cluster. In particular, results for a small number of processors are better in Azure than in the local cluster, however, the results obtained in the local cluster outperforms the ones in Azure when the number of processors grows. Moreover, note that the number of function evaluations required for convergence is larger in the experiments carried out in the local cluster than in the same experiments carried out in Azure when the number of processors is small (10 cores), and it is the opposite for the experiments that use 20 and 40 cores. This behaviour can be attributed to the efficiency of the inter-node communications (remember that each Azure instance has 16 cores). The higher latency in the inter-node communications in Azure leads to a slow propagation of promising results between islands, that results in a slower convergence.

Besides, it is noteworthy that the dispersion of the results is larger for experiments carried out in the Azure public cloud, when the number of cores grows. Figure 5.6 illustrates with hybrid violin/boxplots this fact. As can be seen, the



(a) SSP



(b) HePG2

Figure 5.6: Violin/boxplots comparing results in terms of execution time in Azure vs local cluster (LC).

Table 5.5: Cost evaluation.

problem	**#cores**	**time±std(s)**	**mean price**
SSP	10	3153.95 ± 948.41	1.99 EUR
	20	3057.12 ± 1177.64	1.93 EUR
	40	1861.63 ± 1426.08	1.18 EUR
HePG2	10	151939.74 ± 128256.27	96.10 EUR
	20	87802.45 ± 58619.36	55.53 EUR
	40	68214.13 ± 77442.20	43.14 EUR

number of outliers increases with the number of cores in Azure. Notice that this is exactly the opposite behavior than in the local cluster, and it can be explained by the virtualization overhead in Azure and the use of non-dedicated resources in a multi-tenant platform.

To conclude this evaluation we have found it interesting to carry out a brief study on the cost of these experiments in the Azure public cloud. Conducting a cost analysis comparing the cost of relying on cloud computing and that of owning an in-house cluster would be of particular interest, although is a very difficult task [237]. The acquisition and operational expenses have to be used in estimating the local clusters' cost. However, the actual cost of local clusters is related to its utilization level. For a local cluster acquired as one unit and maintained for several years, the higher the actual utilization level, the lower the effective cost rate. Besides, labor cost in management and maintenance should also be included, which could be significant. Thus, we found unfeasible an accurate estimation of the cost per hour in our local cluster. Besides, if we take a look to the price of the used instances, we can see that in February 2017 the cost of each A9-instance is 2.2769 EUR/hour. The mean pricing for each experiment is shown in Table 5.5. In the view of the obtained results we can conclude that, though our experiments in the cloud demonstrates a slightly poorer performance, in terms of execution time, the cloud *pay-as-you-go* model can be potentially a cost-effective and timely solution for the needs of many users.

5.4. Concluding remarks

In this chapter, we present an evaluation in a public cloud infrastructure of the different parallel algorithms previously proposed in this Thesis. First, we explore and compare the performance of a parameter estimation problem in computational systems biology using the asynPDE implementation proposed in Chapter 2, that is HPC oriented, and a Spark-based implementation, which is throughput oriented. We have assessed both implementations in two different infrastructures: a local cluster and the Microsoft Azure public cloud. Results show that, as was expected, from a computational point of view the MPI implementation outperforms the Spark in terms of execution time. This is mainly due to its low level programming language and reduced overhead. Nevertheless, the Spark implementation should be positively considered since it allows easier programmability and because it also presents further advantages, such as inherent support to node failures and data replication.

Then, we have also evaluated the performance of saCeSS method, proposed in Chapter 4, in the cloud. Unfortunately, we do not know any cloud-based implementation of SS method to compared with.

Although this research was designed and tested with a focus on the field of parameter estimation problems in computational systems biology, we believe that the results obtained in this work can be useful for those researchers interested in the performance of existing traditional parallel metaheuristics in new cloud platforms, as well as in those interested in the potential of new programming models for developing parallel metaheuristic methods.

The results of this chapter have been published in (or submitted to):

- P. González, X. C. Pardo, D. R. Penas, D. Teijeiro, J. R. Banga, and R. Doallo. Using the Cloud for parameter estimation problems: comparing Spark vs MPI with a case-study. *Workshop on Clusters, Clouds and Grids for Life Sciences*, in conjunction with CCGrid 2017 Conference. [82]
- D. Teijeiro, X. C. Pardo, D. R. Penas, P. González, J. R. Banga, and R. Doallo. A cloud-based enhanced differential evolution algorithm for parameter estimation problems in computational systems biology. *Cluster Computing*, under minor revision. [201]

- D. R. Penas, D. Henriques, P. González, R. Doallo, J. Saez-Rodriguez, and J. R. Banga. A parallel metaheuristic for large mixed-integer nonlinear dynamic optimization problems, with applications in computational biology. *PLOS ONE*, under review. [168]

The source code of the two DE implementations used in this work are publicly available at:

- eSiPDE: `https://bitbucket.org/xcpardo/sipde`
- asynPDE: `https://bitbucket.org/DavidPenas/asynpde`, see also Appendix A.

Conclusions and future work

Global optimization is being increasingly used in engineering and across most basic and applied sciences, such as bioinformatics and computational systems biology. In the case of chemical and biological processes, during the last decade there has been a growing interest in modelling their dynamics, i.e. developing kinetic models which are able to encapsulate the time-varying nature of these systems. As a consequence, many research efforts are now being invested in exploiting those dynamic models by mathematical optimization techniques. Metaheuristics are gaining recognition in this context, however, for most realistic applications they still require excessive computation times. In order to reduce the computational cost of these methods, this Thesis makes the following contributions:

- An improved Differential Evolution algorithm (asynPDE) designed to solve complex problems in computational systems biology. The key idea behind this proposal has been to achieve a proper balance of the exploration abilities of DE and the exploitation abilities of efficient local search. The method improved global search through an asynchronous parallel implementation based on a cooperative island-model. The improved local search is implemented by means of several heuristics (efficient local solver, tabu list, logarithmic search) which exploit the structure of parameter estimation problems in systems biology, the main application area considered here. The improved local search mechanism, combined with the parallel cooperation scheme, allows an adequate balance between exploration and exploitation for the class of problems considered. Convergence time can be reduced by several orders of magnitude when the local search heuristics are included in the DE algorithm. Besides, the asynchronous parallel strategy proposed attains a further reduction in the convergence time through collaboration of the parallel processes, demonstrating also a competitive speedup against the synchronous approaches.

- A novel parallel method based on the Scatter Search method, the so-called self-adaptive parallel Cooperative enhanced Scatter Search (saCeSS) algorithm. The implementation proposed combines a coarse-grained parallelization, focused on stimulating the diversification in the search and the cooperation between different processes, with a fine-grained parallelization, aimed at accelerating the computation by performing separate evaluations in parallel. The main features of this proposal have been: (i) a coarse-grained parallelization using a centralized master-slave approach; (ii) a fine-grained parallelization to perform separate cost-function evaluations in parallel and, thus, accelerate the global search; (ii) a cooperation between processes driven by the quality of the solution; (iv) an asynchronous communication protocol to minimize processes' halts; and (v) the dynamical tune of the islands' settings based on their individual progress.
- An extension of the saCeSS method, called saCeSS2, to successfully solve realistic mixed-integer dynamic optimization (MIDO) problems. To this end, the following features have been included in the new saCeSS2 implementation: (i) an efficient mixed-integer local solver (MISQP), (ii) a novel self-adaption mechanism to avoid convergence stagnation, and (iii) the injection of extra diversity during the adaptation steps.
- An extensive and thorough evaluation of the previous proposals using local clusters and supercomputers. The proposed asynPDE method demonstrates its potential for solving non-linear programming (NLP) problems of small and medium size. Moreover, the excellent performance and scalability of saCeSS method have been illustrated considering a set of very challenging parameter estimation problems in large-scale dynamic models of biological systems. The results show that saCeSS is a robust and efficient method, allowing very significant reduction of computation times with respect to previous state of the art methods. Additionally, saCeSS2 has been applied to a set of very challenging MIDO-MINLP problems obtaining also encouraging results.
- An evaluation in a public cloud infrastructure, the Microsoft Azure public

cloud, of the different proposals in this Thesis. The asynPDE implementation proposed, that is HPC oriented, has been compared with a Spark-based implementation, which is throughput oriented. A detailed discussion on the differences that arise from the inherent features of the programming model used in each implementation has been provided, supported by the assessment of the experimental results. Also, a performance evaluation of the saCeSS2 method applied to MIDO-MINLP problems in the Microsoft Azure public cloud has been also reported.

As extensively reported in this Thesis, metaheuristic algorithms have become established as the solution strategies of choice for a large range of optimization problems. Unfortunately, it is not always easy, or even feasible, to anticipate which one of the numerous algorithms already existent will be most suitable for solving a particular problem. This uncertainty is not only limited to different algorithms on different problem classes. There may even be issues with respect to large variations in algorithm performance over different instances of the same problem. Thus, regarding future work, our research will pay attention to extend the developed methods and implementations to be useful for a broader range of applications. In particular, we will focus on:

- extending the saCeSS and saCeSS2 methods to handle multi-objective optimization problems, since many key problems in systems biology involve the simultaneous optimization of multiple conflicting objectives.
- incorporating additional local solvers, especially for the case of MINLP problems.
- generalizing the idea of self-adaptive cooperative search, already explored in this Thesis, through the concept of *multimethod* optimization in which multiple different search algorithms are performed concurrently, and cooperate between them through information exchange.
- considering challenging applications in the related domain of large-scale design in synthetic biology.

Finally, although this research has been designed and tested with focus on the field of parameter estimation problems in computational systems biology, we believe that the results obtained in this Thesis can be useful for those researchers interested in the performance of parallel metaheuristics for global optimization, whether in local clusters, supercomputers, or new cloud platforms, as well as in those interested in the potential of parallel programming models for developing novel parallel metaheuristic methods.

All the proposals in this Thesis, including both the code and the data files needed to reproduce the results reported, have been made publicly available:

- asynPDE:
  `https://bitbucket.org/DavidPenas/asynpde`
- saCeSS:
  `https://bitbucket.org/DavidPenas/sacess-library`
- saCeSS2:
  `https://doi.org/10.5281/zenodo.290219`

The results of this research work have been published in the following journals and conferences:

- Journal Papers (4):
  - D. R. Penas, J. R. Banga, P. González, and R. Doallo. Enhanced parallel differential evolution algorithm for problems in computational systems biology. *Applied Soft Computing*, 33:86–99, 2015. [165]

    JCR Indexed. Impact Factor (JCR 2015): 2.857. Ranking: Q1 [16/104].
  - D. R. Penas, P. González, J. A. Egea, R. Doallo, and J. R. Banga. Parameter estimation in large-scale systems biology models: a parallel and self-adaptive cooperative strategy. *BMC Bioinformatics*, 18(1):52, 2017. [167]

    JCR Indexed. Impact Factor (JCR 2015): 2.435. Ranking: Q1 [10/56].
  - D. Teijeiro, X. C. Pardo, D. R. Penas, P. González, J. R. Banga, and R. Doallo. A cloud-based enhanced differential evolution algorithm for parameter estimation problems in computational systems biology. *Cluster Computing*, under minor revision. [201]

    JCR Indexed. Impact Factor (JCR 2015): 1.514. Ranking: Q2 [28/105].

- D. R. Penas, D. Henriques, P. González, R. Doallo, J. Saez-Rodriguez, and J. R. Banga. A parallel metaheuristic for large mixed-integer nonlinear dynamic optimization problems, with applications in computational biology. *PLOS ONE*, under review. [168]

  JCR Indexed. Impact Factor (JCR 2015): 3.057. Ranking: Q1 [11/63].

▪ International Conferences (6):

- D. R. Penas, J. R. Banga, P. González, and R. Doallo. A parallel differential evolution algorithm for parameter estimation in dynamic models of biological systems. *Advances in Intelligent Systems and Computing*, 294:173–181, 2014. Proceedings of the 8th International Conference on Practical Applications of Computational Biology & Bioinformatics (PACBB 2014). [164]
- D. R. Penas, P. González, J. A. Egea, J. R. Banga, and R. Doallo. Parallel metaheuristics in computational biology: an asynchronous cooperative enhanced scatter search method. *Procedia Computer Science*, 51:630 – 639, 2015. Proceedings of the International Conference On Computational Science (ICCS 2015). [166]
- D. R. Penas, P. González, R. Doallo, J. A. Egea and J. R. Banga. An asynchronous cooperative search metaheuristic for computational systems biology problems. *Advanced Lecture Course on Computational Systems Biology* (CompSysBio 2015). Poster presentation.
- D. R. Penas, P. González, D. Henriques, J. Saez-Rodriguez, R. Doallo, J. R. Banga. A parallel global optimization method to reverse engineer dynamic models of complex biochemical pathways. *Bioinformatics for Young international researchers Expo: Maastricht-Aachen- Liège* (byteMAL 2016). Poster presentation.
- D. R. Penas, P. González, J. A. Egea, R. Doallo, J. R. Banga. Development of large-scale dynamic models of complex biochemical pathways via high-performance computational optimization. *17th International Conference on Systems Biology* (ICSB 2016). Poster presentation.
- P. González, X. C. Pardo, D. R. Penas, D. Teijeiro, J. R. Banga, and R. Doallo. Using the Cloud for parameter estimation problems: comparing

Spark vs MPI with a case-study. *Workshop on Clusters, Clouds and Grids for Life Sciences*, in conjunction with CCGrid 2017 Conference. [82]

Bibliography

[1] W. Abou-Jaoudé, P. Traynard, P. T. Monteiro, J. Saez-Rodriguez, T. Helikar, D. Thieffry, and C. Chaouiya. Logical modeling and dynamical analysis of cellular networks. *Frontiers in Genetics*, 7(86):94, 2016.

[2] B. S. Adiwijaya, P. I. Barton, and B. Tidor. Biological network design strategies: discovery through dynamic optimization. *Molecular BioSystems*, 2(12):650–659, 2006.

[3] D. Agarwal, S. Karamati, S. Puri, and S. K. Prasad. Towards an mpi-like framework for the azure cloud platform. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 176–185. IEEE, 2014.

[4] E. Alba. *Parallel Metaheuristics: A New Class of Algorithms.* Wiley-Interscience, NJ, USA, 2005.

[5] E. Alba, G. Luque, and S. Nesmachnow. Parallel metaheuristics: recent advances and new trends. *International Transactions in Operational Research*, 20(1):1–48, 2013.

[6] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, J. D. Watson, and A. Grimstone. Molecular biology of the cell (3rd edn). *Trends in Biochemical Sciences*, 20(5):210–210, 1995.

[7] B. B. Aldridge, J. M. Burke, D. A. Lauffenburger, and P. K. Sorger. Physicochemical modelling of cell signalling pathways. *Nature cell biology*, 8(11):1195–1203, 2006.

[8] B. B. Aldridge, J. Saez-Rodriguez, J. L. Muhlich, P. K. Sorger, and D. A. Lauffenburger. Fuzzy logic analysis of kinase pathway crosstalk in TNF/EGF/Insulin-induced signaling. *PLOS Computational Biology*, 5(4):1–13, 2009.

[9] A. Alexandrov, R. Bergmann, S. Ewen, J.-C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl, F. Naumann, M. Peters, A. Rheinländer, M. J. Sax, S. Schelter, M. Höger, K. Tzoumas, and D. Warneke. The stratosphere platform for big data analytics. *The VLDB Journal*, 23(6):939–964, 2014.

[10] L. G. Alexopoulos, J. Saez-Rodriguez, B. D. Cosgrove, D. A. Lauffenburger, and P. K. Sorger. Networks inferred from biochemical data reveal profound differences in toll-like receptor and inflammatory signaling between normal and transformed hepatocytes. *Molecular & Cell Proteomics*, 9(9):1849–1865, 2010.

[11] Amazon Web Services. `https://aws.amazon.com`.

[12] J. M. Andión, M. Arenaz, G. Rodríguez, and J. Tourino. A novel compiler support for automatic parallelization on multicore systems. *Parallel Computing*, 39(9):442–460, 2013.

[13] J. Apolloni, J. García-Nieto, E. Alba, and G. Leguizamón. Empirical evaluation of distributed differential evolution on standard benchmarks. *Applied Mathematics and Computation*, 236(0):351–366, 2014.

[14] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[15] F. Aydemir, A. Günay, F. Öztoprak, S. Birbil, and P. Yolum. Multiagent cooperation for solving global optimization problems: An extendible framework with example cooperation strategies. *Journal of Global Optimization*, 57(2):499–519, 2013.

[16] E. Balsa-Canto, J. Banga, and A. A. Alonso. An optimal identification procedure for model development ins systems biology: Applications in cell signalling. *Foundations of Systems Biology in Engineering*, 4(1):51–56, 2007.

[17] E. Balsa-Canto and J. R. Banga. AMIGO, a toolbox for advanced model identification in systems biology using global optimization. *Bioinformatics*, 27(16):2311–2313, 2011.

[18] E. Balsa-Canto, M. Peifer, J. R. Banga, J. Timmer, and C. Fleck. Hybrid optimization method with general switching strategy for parameter estimation. *BMC Systems Biology*, 2(1):26, 2008.

[19] J. R. Banga. Optimization in computational systems biology. *BMC Systems Biology*, 2(1):47, 2008.

[20] J. R. Banga and E. Balsa-Canto. Parameter estimation and optimal experimental design. *Essays in Biochemistry*, 45:195–210, 2008.

[21] J. R. Banga, E. Balsa-Canto, C. G. Moles, and A. A. Alonso. Dynamic optimization of bioprocesses: Efficient and robust numerical strategies. *Journal of Biotechnology*, 117(4):407–419, 2005.

[22] K. Becker, E. Balsa-Canto, D. Cicin-Sain, A. Hoermann, H. Janssens, J. R. Banga, and J. Jaeger. Reverse-engineering post-transcriptional regulation of gap genes in drosophila melanogaster. *PLOS Computational Biology*, 9(10):e1003281, 2013.

[23] R. Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences*, 42(10):767–769, 1956.

[24] R. Bellman. *Dynamic programming.* Princeton University Press, Princeton N.J, 1957.

[25] M. Bernardo-Faura, S. Massen, C. S. Falk, N. R. Brady, and R. Eils. Data-derived modeling characterizes plasticity of mapk signaling in melanoma. *PLOS Computational Biology*, 10(9):1–18, 2014.

[26] D. P. Bertsekas. *Dynamic programming and optimal control.* Athena Scientific Belmont, MA, 1995.

[27] H.-G. Beyer and H.-P. Schwefel. Evolution strategies – a comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

[28] L. T. Biegler, A. M. Cervantes, and A. Wächter. Advances in simultaneous strategies for dynamic process optimization. *Chemical Engineering Science*, 57(4):575–593, 2002.

[29] R. Bonneau, D. J. Reiss, P. Shannon, M. Facciotti, L. Hood, N. S. Baliga, and V. Thorsson. The inferelator: an algorithm for learning parsimonious regulatory networks from systems-biology data sets *de novo*. *Genome Biology*, 7:R36, 2006.

[30] F. Boukouvala, R. Misener, and C. A. Floudas. Global optimization advances in mixed-integer nonlinear programming, minlp, and constrained derivative-free optimization, cdfo. *European Journal of Operational Research*, 252(3):701–727, 2016.

[31] I. Boussaïd, J. Lepagnot, and P. Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237(1):82–117, 2013.

[32] W. Bożejko and M. Wodecki. Parallel path-relinking method for the flow shop scheduling problem. In *International Conference on Computational Science*, pages 264–273. Springer, 2008.

[33] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *Evolutionary Computation, IEEE Transactions on*, 10(6):646–657, 2006.

[34] A. E. Bryson. *Applied optimal control: optimization, estimation and control.* CRC Press, 1975.

[35] S. Burer and A. N. Letchford. Non-convex mixed-integer nonlinear programming: A survey. In *Surveys in Operations Research and Management Science*, pages 97–106, 2012.

[36] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In *Handbook of metaheuristics*, pages 457–474. Springer, 2003.

[37] K. P. Burnham and D. R. Anderson. *Model selection and multimodel inference: a practical information-theoretic approach.* Springer Science & Business Media, 2003.

[38] R. Buyya, J. Broberg, and A. M. Goscinski. *Cloud computing: Principles and paradigms*, volume 87. John Wiley & Sons, 2010.

[39] S. Campanoni, T. M. Jones, G. Holloway, G.-Y. Wei, and D. Brooks. Helix: Making the extraction of thread-level parallelism mainstream. *IEEE Micro*, 32(4):8–18, 2012.

[40] CESGA. Svg specifications. `https://www.cesga.es/gl/infraestructuras/computacion/svg`.

[41] B. Chachuat, A. Singer, and P. Barton. Global methods for dynamic optimization and mixed-integer dynamic optimization. *Industrial & Engineering Chemistry Research*, 45(25):8373–8392, 2006.

[42] U. K. Chakraborty. *Advances in Differential Evolution.* Springer, 2008.

[43] Chapel Parallel Programming Language. `http://chapel.cray.com`.

[44] C. L. Chen and H. W. Tsai. Model-based insulin therapy scheduling: A mixed-integer nonlinear dynamic optimization approach. *Industrial & Engineering Chemistry Research*, 48(18):8595–8604, 2009.

[45] T. G. Crainic and M. Toulouse. *Parallel Strategies for Meta-Heuristics*, pages 475–513. Springer, 2003.

[46] T. G. Crainic, M. Toulouse, and M. Gendreau. Toward a taxonomy of parallel tabu search heuristics. *INFORMS Journal on Computing*, 9(1):61–72, 1997.

[47] S. Da Ros, G. Colusso, T. Weschenfelder, L. De Marsillac Terra, F. De Castilhos, M. Corazza, and M. Schwaab. A comparison among stochastic optimization algorithms for parameter estimation of biochemical kinetic models. *Applied Soft Computing Journal*, 13(5):2205–2214, 2013.

[48] S. Das and P. Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 15(1):4–31, 2011.

[49] M. S. Dasika and C. D. Maranas. Optcircuit: an optimization based method for computational design of genetic circuits. *BMC Systems Biology*, 2(1):24, 2008.

[50] G. M. de Hijas-Liste, E. Klipp, E. Balsa-Canto, and J. R. Banga. Global dynamic optimization approach to predict activation in metabolic pathways. *BMC Systems Biology*, 8(1):1, 2014.

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

[52] M. de la Maza and D. Yuret. Dynamic hill climbing. *AI Expert*, 9(3):26–31, 1994.

[53] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communication ACM*, 51(1):107–113, 2008.

[54] J. E. Dennis, Jr., D. M. Gay, and R. E. Welsch. Algorithm 573: Nl2sol - an adaptive nonlinear least-squares algorithm. *ACM Transactions on Mathematical Software (TOMS)*, 7(3):369–383, 1981.

[55] J. Derrac, S. García, D. Molina, and F. Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011.

[56] F. J. Doyle and J. Stelling. Systems interface biology. *Journal of the Royal Society Interface*, 3(10):603–616, 2006.

[57] P. Dua, V. Dua, and E. N. Pistikopoulos. Optimal delivery of chemotherapeutic agents in cancer. *Computers & Chemical Engineering*, 32(1):99–107, 2008.

[58] O. J. Dunn. Multiple comparisons using rank sums. *Technometrics*, 6(3):241–252, 1964.

[59] J. A. Egea. *New heuristics for global optimization of complex bioprocesses*. PhD thesis, University of Vigo, 2008.

[60] J. A. Egea, E. Balsa-Canto, M.-S. G. García, and J. R. Banga. Dynamic optimization of nonlinear processes with an enhanced scatter search method. *Industrial & Engineering Chemistry Research*, 48(9):4388–4401, 2009.

[61] J. A. Egea, R. Martí, and J. R. Banga. An evolutionary method for complex-process optimization. *Computers & Operations Research*, 37(2):315–324, 2010.

[62] J. A. Egea, M. Rodríguez-Fernández, J. R. Banga, and R. Martí. Scatter search for chemical and bio-process optimization. *Journal of Global Optimization*, 37(3):481–503, 2007.

[63] J. Ekanayake and G. Fox. High performance parallel computing with clouds and cloud technologies. In *International Conference on Cloud Computing*, pages 20–38. Springer, 2009.

[64] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox. Twister: A runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 810–818. ACM, 2010.

[65] European Bioinformatics Institute (EMBL-EBI). `http://www.ebi.ac.uk`.

[66] C. Evangelinos and C. Hill. Cloud computing for parallel scientific HPC applications: Feasibility of running coupled atmosphere-ocean climate models on amazon's EC2. In *1st Workshop on Cloud Computing and its Applications (CCA'08)*, pages 1–6, 2008.

[67] O. Exler, L. T. Antelo, J. A. Egea, A. A. Alonso, and J. R. Banga. A tabu search-based algorithm for mixed-integer nonlinear problems and its application to integrated process and control system design. *Computers & Chemical Engineering*, 32(8):1877–1891, 2008.

[68] O. Exler, T. Lehmann, and K. Schittkowski. A comparative study of sqp-type algorithms for nonlinear and nonconvex mixed-integer optimization. *Mathematical Programming Computation*, 4(4):383–412, 2012.

[69] O. Exler and K. Schittkowski. A trust region sqp algorithm for mixed-integer nonlinear programming. *Optimization Letters*, 1(3):269–280, 2007.

[70] R. R. Expósito, G. L. Taboada, S. Ramos, J. Touriño, and R. Doallo. Performance analysis of HPC applications in the cloud. *Future Generation Computer Systems*, 29(1):218–229, 2013.

[71] M. Fan, H. Kuwahara, X. Wang, S. Wang, and X. Gao. Parameter estimation methods for gene circuit modeling from time-series mrna data: a comparative study. *Briefings in Bioinformatics*, 16(6):987–999, 2015.

[72] A. Flores-Tlacuahuac and L. T. Biegler. Simultaneous mixed-integer dynamic optimization for integrated design and control. *Computers & chemical engineering*, 31(5):588–600, 2007.

[73] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21(9):948–960, 1972.

[74] A. Gábor and J. R. Banga. Robust and efficient parameter estimation in dynamic models of biological systems. *BMC Systems Biology*, 9(1):74, 2015.

[75] K. G. Gadkar, R. Gunawan, and F. J. Doyle. Iterative approach to model identification of biological networks. *BMC Bioinformatics*, 6(1):155, 2005.

[76] F. Garcıa-López, B. Melián-Batista, J. A. Moreno-Pérez, and J. M. Moreno-Vega. Parallelization of the scatter search for the p-median problem. *Parallel Computing*, 29(5):575–589, 2003.

[77] Gartner, Inc. `http://www.gartner.com`.

[78] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine. A users' guide and tutorial for networked parallel computing.* MIT press, 1994.

[79] Global Arrays. `http://hpc.pnl.gov/globalarrays/`.

[80] F. Glover. A template for scatter search and path relinking. In *Selected Papers from the Third European Conference on Artificial Evolution*, pages 3–54. Springer, 1998.

[81] F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 29(3):652–684, 2000.

[82] P. González, X. C. Pardo, D. R. Penas, D. Teijeiro, J. R. Banga, and R. Doallo. Using the cloud for parameter estimation problems: comparing spark vs mpi with a case-study (accepted). In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID'2017)*, 2017.

[83] J.-P. Goux and S. Leyffer. Solving large minlps on computational grids. *Optimization and Engineering*, 3(3):327–346, 2002.

[84] H. Greenberg, W. Hart, and G. Lancia. Opportunities for combinatorial optimization in computational biology. *INFORMS Journal on Computing*, 16(3):211–231, 2004.

[85] G. Guillén-Gosálbez, A. Miró, R. Alves, A. Sorribas, and L. Jiménez. Identification of regulatory structure and kinetic parameters of biochemical networks via mixed-integer dynamic optimization. *BMC Systems Biology*, 7(1):113, 2013.

[86] G. Hager and G. Wellein. *Introduction to high performance computing for scientists and engineers*. CRC Press, 1st edition, 2010.

[87] N. Hansen, A. Auger, S. Finck, and R. Ros. Real-parameter black-box optimization benchmarking 2010: Experimental setup. Technical Report Rapports de Recherche RR-6828, Institut National de Recherche en Informatique et en Automatique (INRIA), 2009.

[88] J. Hasenauer, S. Waldherr, K. Wagner, and F. Allgower. Parameter identification, experimental design and model falsification for biological network models using semidefinite programming. *IET Systems Biology*, 4(2):119–130, 2010.

[89] V. Hatzimanikatis, C. A. Floudas, and J. E. Bailey. Analysis and design of metabolic reaction networks via mixed-integer linear optimization. *AIChE Journal*, 42(5):1277–1292, 1996.

[90] V. Hatzimanikatis, C. A. Floudas, and J. E. Bailey. Optimization of regulatory architectures in metabolic reaction networks. *Biotechnology and Bioengineering*, 52(4):485–500, 1996.

[91] S. Hazelhurst. Scientific computing using virtual high-performance computing: a case study using the amazon elastic computing cloud. In *Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology*, pages 94–103. ACM, 2008.

[92] T. Helikar, J. Konvalina, J. Heidel, and J. A. Rogers. Emergent decision-making in biological signal transduction networks. *Proceedings of the National Academy of Sciences*, 105(6):1913–1918, 2008.

[93] J. L. Hennessy and D. A. Patterson. *Computer Architecture, Fifth Edition: A Quantitative Approach.* Morgan Kaufmann Publishers Inc., 5th edition, 2011.

[94] D. Henriques, M. Rocha, J. Saez-Rodriguez, and J. R. Banga. Reverse engineering of logic-based differential equation models using a mixed-integer dynamic optimization approach. *Bioinformatics*, 31(18):2999–3007, 2015.

[95] D. Henriques, A. F. Villaverde, M. Rocha, J. Saez-Rodriguez, and J. R. Banga. Data-driven reverse engineering of signaling pathways using ensembles of dynamic models. *PLOS Computational Biology*, 13(2):e1005379, 2017.

[96] S. M. Hill, L. M. Heiser, T. Cokelaer, M. Unger, N. K. Nesser, D. E. Carlin, Y. Zhang, A. Sokolov, E. O. Paull, C. K. Wong, et al. Inferring causal molecular networks: empirical assessment through a community-based effort. *Nature methods*, 13(4):310–318, 2016.

[97] A. Hindmarsh, P. Brown, K. Grant, S. Lee, R. Serban, D. Shumaker, and C. Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software*, 31(3):363–396, 2005.

[98] T. Hirmajer, E. Balsa-Canto, and J. R. Banga. Dotcvpsb, a software toolbox for dynamic optimization in systems biology. *BMC Bioinformatics*, 10(1):199, 2009.

[99] T. Hirmajer, E. Balsa-Canto, and J. R. Banga. Mixed-integer non-linear optimal control in systems biology and biotechnology: numerical methods and a software toolbox. *IFAC Proceedings Volumes*, 43(5):314–319, 2010.

[100] K. Hwang, J. Dongarra, and G. C. Fox. *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*. Morgan Kaufmann, 1st edition, 2013.

[101] D. Izzo, M. Rucinski, and C. Ampatzis. Parallel global optimisation metaheuristics using an asynchronous island-model. In *Proceedings of the Eleventh Conference on Congress on Evolutionary Computation*, pages 2301–2308. IEEE, 2009.

[102] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright. Performance analysis of high performance computing applications on the amazon web services cloud. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 159–168. IEEE, 2010.

[103] J. Jaeger and N. A. Monk. Reverse engineering of gene regulatory networks. *Learning and Inference in Computational Systems Biology*, 0(0):9–34, 2010.

[104] G. Jia, G. Stephanopoulos, and R. Gunawan. Incremental parameter estimation of kinetic metabolic network models. *BMC Systems Biology*, 6(1):1, 2012.

[105] C. Jin, C. Vecchiola, and R. Buyya. MRPGA: an extension of MapReduce for parallelizing genetic algorithms. In *IEEE Fourth International Conference on eScience, eScience'08*, pages 214–221. IEEE, 2008.

[106] H. Jin, D. Jespersen, P. Mehrotra, R. Biswas, L. Huang, and B. Chapman. High performance computing using mpi and openmp on multi-core parallel systems. *Parallel Computing*, 37(9):562–575, 2011.

[107] C. Jørgensen and R. Linding. Simplistic pathways or complex networks? *Current Opinion in Genetics and Development*, 20(1):15–22, 2010.

[108] L. Jostins and J. Jaeger. Reverse engineering a gene network using an asynchronous parallel evolution strategy. *BMC Systems Biology*, 4(1):17, 2010.

[109] A. Karbowski, M. Majchrowski, and P. Trojanek. jPar–a simple, free and lightweight tool for parallelizing Matlab calculations on multicores and in

clusters. In *9th International Workshop on State-of-the-Art in Scientific and Parallel Computing (PARA 2008)*, pages 91–96, 2008.

[110] Karl Rupp, Freelance Computational Scientist . `https://www.karlrupp.net/2015/06/40-years-of-microprocessor-trend-data`.

[111] S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3):437–467, 1969.

[112] K. Keahey, T. Freeman, J. Lauret, and D. Olson. Virtual workspaces for scientific applications. *Journal of Physics: Conference Series*, 78(1):012038, 2007.

[113] A. Kiparissides, M. Koutinas, C. Kontoravdi, A. Mantalaris, and E. N. Pistikopoulos. 'closing the loop' in biological systems modeling - from the in silico to the in vitro. *Automatica*, 47(6):1147–1155, 2011.

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

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

[116] A. Kremling and J. Saez-Rodriguez. Systems biology—an engineering perspective. *Journal of Biotechnology*, 129(2):329–351, 2007.

[117] J. Krumsiek, S. Poelsterl, D. M. Wittmann, and F. J. Theis. Odefy - from discrete to continuous models. *BMC Bioinformatics*, 11(1):233, 2010.

[118] W. H. Kruskal and W. A. Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):583–621, 1952.

[119] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to parallel computing: design and analysis of algorithms*, volume 400. Benjamin/Cummings Redwood City, CA, 1994.

[120] J.-I. Kushida, K. Oba, A. Hara, and T. Takahama. Solving quadratic assignment problems by differential evolution. In *Soft Computing and Intelligent Systems (SCIS) and 13th International Symposium on Advanced Intelligent Systems (ISIS), 2012 Joint 6th International Conference on*, pages 639–644. IEEE, 2012.

[121] P. Larrañaga, B. Calvo, R. Santana, C. Bielza, J. Galdiano, I. Inza, J. Lozano, R. Armañanzas, G. Santafé, A. Pérez, and V. Robles. Machine learning in bioinformatics. *Briefings in Bioinformatics*, 7(1):86–112, 2006.

[122] D. Lebiedz, S. Sager, H. Bock, and P. Lebiedz. Annihilation of limit-cycle oscillations by identification of critical perturbing stimuli via mixed-integer optimal control. *Physical Review Letters*, 95(10):108303, 2005.

[123] W. Lee, Y. Hsiao, and W. Hwang. Designing a parallel evolutionary algorithm for inferring gene networks on the cloud computing environment. *BMC Systems Biology*, 8(1):5, 2014.

[124] J. Li, M. Humphrey, C. Van Ingen, D. Agarwal, K. Jackson, and Y. Ryu. escience in the cloud: A modis satellite data reprojection and reduction pipeline in the windows azure platform. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–10. IEEE, 2010.

[125] D. Liberzon. *Calculus of variations and optimal control theory: a concise introduction*. Princeton University Press, 2012.

[126] T. Lipniacki, P. Paszek, A. Brasier, B. Luxon, and M. Kimmel. Mathematical model of nf-κb regulatory module. *Journal of Theoretical Biology*, 228(2):195–215, 2004.

[127] J. Locke, A. Millar, and M. Turner. Modelling genetic networks with noisy and varied experimental data: The circadian clock in arabidopsis thaliana. *Journal of Theoretical Biology*, 234(3):383–393, 2005.

[128] F. G. López, M. G. Torres, B. M. Batista, J. A. M. Pérez, and J. M. Moreno-Vega. Solving feature subset selection problem by a parallel scatter search. *European Journal of Operational Research*, 169(2):477–489, 2006.

[129] X. Lu, F. Liang, B. Wang, L. Zha, and Z. Xu. DataMPI: extending MPI to Hadoop-like big data computing. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 829–838. IEEE, 2014.

[130] D. G. Luenberger and Y. Ye. *Linear and nonlinear programming*, volume 228. Springer, 2015.

[131] S. Luke. *Essentials of Metaheuristics.* Lulu, second edition, 2013.

[132] A. MacNamara, D. Henriques, and J. Saez-Rodriguez. *Modeling Signaling Networks with Different Formalisms: A Preview*, pages 89–105. Humana Press, 2013.

[133] A. MacNamara, C. Terfve, D. Henriques, B. P. Bernabé, and J. Saez-Rodriguez. State–time spectrum of signal transduction logic models. *Physical Biology*, 9(4):045003, 2012.

[134] R. Mahadevan, J. S. Edwards, and F. J. Doyle. Dynamic flux balance analysis of diauxic growth in *Escherichia coli*. *Biophysical journal*, 83(3):1331–1340, 2002.

[135] A. W. McNabb, C. K. Monson, and K. D. Seppi. Parallel PSO using MapReduce. In *IEEE Congress on Evolutionary Computation, CEC2007*, pages 7–14. IEEE, 2007.

[136] P. Mendes and D. Kell. Non-linear optimization of biochemical pathways: applications to metabolic engineering and parameter estimation. *Bioinformatics*, 14(10):869–883, 1998.

[137] L. Mendoza and I. Xenarios. A method for the generation of standardized qualitative dynamical systems of regulatory networks. *Theoretical Biology and Medical Modelling*, 3(1):13, 2006.

[138] F. Menolascina, V. Siciliano, and D. Di Bernardo. Engineering and control of biological systems: a new way to tackle complex diseases. *FEBS letters*, 586(15):2122–2128, 2012.

[139] Microsoft Azure. `https://azure.microsoft.com`.

[140] M. Mohideen, J. Perkins, and E. Pistikopoulos. Towards an efficient numerical procedure for mixed integer optimal control. *Computers & Chemical Engineering*, 21:S457–S462, 1997.

[141] C. G. Moles, P. Mendes, and J. R. Banga. Parameter estimation in biochemical pathways: a comparison of global optimization methods. *Genome Research*, 13(11):2467–2474, 2003.

[142] G. E. Moore. Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff. *IEEE Solid-State Circuits Society Newsletter*, 11(5):33–35, 2006.

[143] M. K. Morris, J. Saez-Rodriguez, D. C. Clarke, P. K. Sorger, and D. A. Lauffenburger. Training signaling pathway maps to biochemical data with constrained fuzzy logic: quantitative analysis of liver cell responses to inflammatory stimuli. *PLOS Computational Biology*, 7(3):1–20, 2011.

[144] M. K. Morris, J. Saez-Rodriguez, P. K. Sorger, and D. A. Lauffenburger. Logic-based models for the analysis of cell signaling networks. *Biochemistry*, 49(15):3216–3224, 2010.

[145] MPI Forum, Standardization Forum for the Message Passing Interface (MPI). `http://mpi-forum.org`.

[146] A. Munawar, M. Wahib, M. Munetomo, and K. Akama. Advanced genetic algorithm to solve minlp problems over gpu. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 318–325, 2011.

[147] J. Napper and P. Bientinesi. Can cloud computing reach the top500? In *Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop*, pages 17–20. ACM, 2009.

[148] F. Neri, C. Cotta, and P. Moscato. *Handbook of memetic algorithms*, volume 379. Springer, 2012.

[149] F. Neri and V. Tirronen. On memetic differential evolution frameworks: a study of advantages and limitations in hybridization. In *IEEE Congress on Evolutionary Computation, 2008. CEC 2008.*, pages 2135–2142, 2008.

[150] F. Neri and V. Tirronen. Recent advances in differential evolution: A survey and experimental analysis. *Artificial Intelligence Review*, 33(1-2):61–106, 2010.

[151] F. Neri, V. Tirronen, and T. Kärkkäinen. Enhancing differential evolution frameworks by scale factor local search - part ii. In *2009 IEEE Congress on Evolutionary Computation, CEC 2009*, pages 118–125, 2009.

[152] N. Noman and H. Iba. Enhancing differential evolution performance with local search for high dimensional function optimization. In *GECCO 2005 - Genetic and Evolutionary Computation Conference*, pages 967–974, 2005.

[153] N. Noman and H. Iba. Accelerating differential evolution using an adaptive local search. *IEEE Transactions on Evolutionary Computation*, 12(1):107–125, 2008.

[154] M. S. Ntipteni, I. M. Valakos, and I. K. Nikolos. An asynchronous parallel differential evolution algorithm. In *Proceedings of the ERCOFTAC Conference on Design Optimisation: Methods and Application*, 2006.

[155] J. Olenšek, T. Tuma, J. Puhan, and Árpád Bűrmen. A new asynchronous parallel global optimization method based on simulated annealing and differential evolution. *Applied Soft Computing*, 11(1):1481 – 1489, 2011.

[156] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. An early performance analysis of cloud computing services for scientific computing. Technical Report PDS-2008-006, Delft University of Technology, 2008.

[157] R. Östermark. Solving difficult mixed integer and disjunctive non-linear problems on single and parallel processors. *Applied Soft Computing*, 24:385–405, 2014.

[158] I. Otero-Muras and J. R. Banga. Multicriteria global optimization for biocircuit design. *BMC Systems Biology*, 8(1):113, 2014.

[159] I. Otero-Muras and J. R. Banga. Design principles of biological oscillators through optimization: Forward and reverse analysis. *PLOS ONE*, 11(12):1–26, 2016.

[160] I. Otero-Muras, D. Henriques, and J. R. Banga. Synbadm: a tool for optimization-based automated design of synthetic gene circuits. *Bioinformatics*, 32(21):3360–3362, 2016.

[161] P. S. Pacheco. *Parallel programming with MPI.* Morgan Kaufmann, 1996.

[162] M. Palkowski, T. Klimek, and W. Bielecki. Traco: An automatic loop nest parallelizer for numerical applications. In *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 681–686. IEEE, 2015.

[163] E. T. Papoutsakis. Equations and calculations for fermentations of butyric acid bacteria. *Biotechnology and Bioengineering*, 26(2):174–187, 1984.

[164] D. R. Penas, J. R. Banga, P. González, and R. Doallo. A parallel differential evolution algorithm for parameter estimation in dynamic models of biological systems. In *8th International Conference on Practical Applications of Computational Biology & Bioinformatics (PACBB 2014)*, volume 294, pages 173–181. Springer, 2014.

[165] D. R. Penas, J. R. Banga, P. González, and R. Doallo. Enhanced parallel differential evolution algorithm for problems in computational systems biology. *Applied Soft Computing*, 33(0):86–99, 2015.

[166] D. R. Penas, P. González, J. A. Egea, J. R. Banga, and R. Doallo. Parallel metaheuristics in computational biology: An asynchronous cooperative enhanced scatter search method. *Procedia Computer Science*, 51(0):630 – 639, 2015. International Conference On Computational Science (ICCS 2015).

[167] D. R. Penas, P. González, J. A. Egea, R. Doallo, and J. R. Banga. Parameter estimation in large-scale systems biology models: a parallel and self-adaptive cooperative strategy. *BMC Bioinformatics*, 18(1):52, 2017.

[168] D. R. Penas, D. Henriques, P. González, R. Doallo, J. Saez-Rodriguez, and J. R. Banga. A parallel metaheuristic for large mixed-integer nonlinear dynamic optimization problems, with applications in computational biology. *PLOS ONE, under review*, 2017.

[169] T. J. Perkins, J. Jaeger, J. Reinitz, and L. Glass. Reverse engineering the gap gene network of drosophila melanogaster. *PLOS Computational Biology*, 2(5):1–12, 2006.

[170] PORT library. `http://www.netlib.org/port/`.

[171] M. Powell. Convergence properties of a class of minimization algorithms. *Nonlinear Programming*, 2(0):1–27, 1975.

[172] K. Price, R. M. Storn, and J. A. Lampinen. *Differential evolution: a practical approach to global optimization.* Springer Science & Business Media, 2006.

[173] N. D. Price, J. L. Reed, and B. Ø. Palsson. Genome-scale models of microbial cells: evaluating the consequences of constraints. *Nature Reviews Microbiology*, 2(11):886–897, 2004.

[174] R. J. Prill, J. Saez-Rodriguez, L. G. Alexopoulos, P. K. Sorger, and G. Stolovitzky. Crowdsourcing network inference: The dream predictive signaling network challenge. *Science Signaling*, 4(189):mr7, 2011.

[175] G. Quaranta, G. Marano, R. Greco, and G. Monti. Parametric identification of seismic isolators using differential evolution and particle swarm optimization. *Applied Soft Computing Journal*, 22(0):458–464, 2014.

[176] A. Radenski. Distributed simulated annealing with MapReduce. In *Proceedings of the 2012T European Conference on Applications of Evolutionary Computation*, pages 466–476. Springer, 2012.

[177] L. Ramakrishnan, K. R. Jackson, S. Canon, S. Cholia, and J. Shalf. Defining future platform requirements for e-science clouds. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 101–106. ACM, 2010.

[178] D. A. Reed and J. Dongarra. Exascale computing and big data. *Communications of the ACM*, 58(7):56–68, 2015.

[179] S. Reinker, R. Altman, and J. Timmer. Parameter estimation in stochastic biochemical reactions. *IEE Proceedings-Systems Biology*, 153(4):168–178, 2006.

[180] I. Rodero, M. Parashar, A. Quiroz, F. Guim, and S. W. Poole. *Handbook of Energy-Aware and Green Computing-Two Volume Set*, chapter Energy-efficient Online Provisioning for HPC Workloads, pages 795–816. Chapman and Hall/CRC 2012, 2012.

[181] M. Rodriguez-Fernandez, J. A. Egea, and J. R. Banga. Novel metaheuristic for parameter estimation in nonlinear dynamic biological systems. *BMC Bioinformatics*, 7(1):483, 2006.

[182] M. Ruciński, D. Izzo, and F. Biscani. On the impact of the migration topology on the island model. *Parallel Computing*, 36(10-11):555–571, 2010.

[183] J. Saez-Rodriguez, L. G. Alexopoulos, J. Epperlein, R. Samaga, D. A. Lauffenburger, S. Klamt, and P. K. Sorger. Discrete logic modelling as a means to link protein signalling networks with functional analysis of mammalian signal transduction. *Molecular Systems Biology*, 5:331, 2009.

[184] J. Saez-Rodriguez, J. C. Costello, S. H. Friend, M. R. Kellen, L. Mangravite, P. Meyer, T. Norman, and G. Stolovitzky. Crowdsourcing biomedical research: leveraging communities as innovation engines. *Nature Reviews Genetics*, 17(8):470–486, 2016.

[185] S. Sager. A benchmark library of mixed-integer optimal control problems. In *Mixed Integer Nonlinear Programming*, pages 631–670. Springer, 2012.

[186] S. Sager, H. G. Bock, and M. Diehl. The integer approximation error in mixed-integer optimal control. *Mathematical Programming*, 133(1-2):1–23, 2012.

[187] S. Sager, M. Claeys, and F. Messine. Efficient upper and lower bounds for global mixed-integer optimal control. *Journal of Global Optimization*, 61(4):721–743, 2014.

[188] F. Sambo, M. A. Montes de Oca, B. Di Camillo, G. Toffolo, and T. Stutzle. More: Mixed optimization for reverse engineering—an application to modeling biological networks response via sparse systems of nonlinear differential equations. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 9(5):1459–1471, 2012.

[189] M. Schlüter, J. A. Egea, and J. R. Banga. Extended ant colony optimization for non-convex mixed integer nonlinear programming. *Computers & Operations Research*, 36(7):2217–2229, 2009.

[190] M. Schlüter, J. A. Egea, L. T. Antelo, A. A. Alonso, and J. R. Banga. An extended ant colony optimization algorithm for integrated process and control system design. *Industrial & Engineering Chemistry Research*, 48(14):6723–6738, 2009.

[191] E. Schneider and R. A. Krohling. Differential evolution and tabu search to find multiple solutions of multimodal optimization problems. In *Soft Computing*

in Industrial Applications, volume 223, pages 61–69. Springer International Publishing, 2014.

[192] J. Sendín, O. Exler, and J. Banga. Multi-objective mixed integer strategy for the optimisation of biological networks. *IET Systems Biology*, 4(3):236–248, 2010.

[193] M. Srinivas and G. Rangaiah. Differential evolution with tabu list for global optimization and its application to phase equilibrium and parameter estimation problems. *Industrial and Engineering Chemistry Research*, 46(10):3410–3421, 2007.

[194] R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[195] K. Streit, J. Doerfert, C. Hammacher, A. Zeller, and S. Hack. Generalized task parallelism. *ACM Transactions on Architecture and Code Optimization (TACO)*, 12(1):8:1–8:25, 2015.

[196] E.-G. Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

[197] D. Tasoulis, N. Pavlidis, V. Plagianakos, and M. Vrahatis. Parallel differential evolution. In *Proceedings of the 2004 Congress on Evolutionary Computation, CEC2004*, volume 2, pages 2023–2029, 2004.

[198] D. Teijeiro, X. C. Pardo, P. González, J. R. Banga, and R. Doallo. Implementing parallel differential evolution on Spark. In *Applications of Evolutionary Computation. Lecture Notes in Computer Science, Vol. 9598*, pages 75–90. Springer, 2016.

[199] D. Teijeiro, X. C. Pardo, P. González, J. R. Banga, and R. Doallo. Towards cloud-based parallel metaheuristics: A case study in computational biology with differential evolution and spark. *International Journal of High Performance Computing Applications*, 0(0):1094342016679011, 2016.

[200] D. Teijeiro, X. C. Pardo, D. R. Penas, P. González, J. R. Banga, and R. Doallo. Evaluation of parallel differential evolution implementations on mapreduce and spark. In *Lecture Notes in Computer Science, in press.* Springer, 2016.

[201] D. Teijeiro, X. C. Pardo, D. R. Penas, P. González, J. R. Banga, and R. Doallo. A cloud-based enhanced differential evolution algorithm for parameter estimation problems in computational systems biology. *Cluster Computing , under minor revision*, 2017.

[202] C. Terfve, T. Cokelaer, D. Henriques, A. MacNamara, E. Goncalves, M. K. Morris, M. van Iersel, D. A. Lauffenburger, and J. Saez-Rodriguez. CellNOptR: a flexible toolkit to train protein signaling networks to data using multiple logic formalisms. *BMC Systems Biology*, 6(1):133, 2012.

[203] The OpenMP API specification for parallel programming. `http://www.openmp.org`.

[204] R. Thomas, C. J. Paredes, S. Mehrotra, V. Hatzimanikatis, and E. T. Papoutsakis. A model-based optimization framework for the inference of regulatory interactions using time-course dna microarray expression data. *BMC Bioinformatics*, 8(1):228, 2007.

[205] V. Tirronen, F. Neri, and T. Rossi. Enhancing differential evolution frameworks by scale factor local search - part i. In *2009 IEEE Congress on Evolutionary Computation, CEC 2009*, pages 94–101, 2009.

[206] Top500. `https://www.top500.org`.

[207] M. Toulouse, T. G. Crainic, and M. Gendreau. Communication issues in designing cooperative multi-thread parallel searches. In I. Osman and J. Kelly, editors, *Meta-Heuristics*, pages 503–522. Springer, 1996.

[208] M. Toulouse, T. G. Crainic, and B. Sansó. Systemic behavior of cooperative search algorithms. *Parallel Computing*, 30(1):57–79, 2004.

[209] M. Toulouse, T. G. Crainic, and K. Thulasiraman. Global optimization properties of parallel cooperative search algorithms: a simulation study. *Parallel Computing*, 26(1):91–112, 2000.

[210] Unified Parallel C. `http://upc.gwu.edu`.

[211] H. Vandierendonck, S. Rul, and K. De Bosschere. The paralax infrastructure: automatic parallelization with a helping hand. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, pages 389–400. ACM, 2010.

[212] A. Varma and B. O. Palsson. Metabolic flux balancing: Basic concepts, scientific and practical use. *Nature Biotechnology*, 12(1):994–998, 1994.

[213] V. S. Vassiliadis. *Computational Solution of Dynamic Optimization Problems with General Differential-Algebraic Constraints.* PhD thesis, University of London, London, U.K., 1993.

[214] V. S. Vassiliadis, R. Sargent, and C. Pantelides. Solution of a class of multi-stage dynamic optimization problems. 1. problems without path constraints. *Industrial & Engineering Chemistry Research*, 33(9):2111–2122, 1994.

[215] M. Črepinšek, S.-H. Liu, and M. Mernik. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys*, 45(3):35:1–35:33, 2013.

[216] M. Črepinšek, S.-H. Liu, and M. Mernik. Replication and comparison of computational experiments in applied evolutionary computing: Common pitfalls and guidelines to avoid them. *Applied Soft Computing*, 19(0):161–170, 2014.

[217] A. Verma, X. Llora, D. E. Goldberg, and R. H. Campbell. Scaling genetic algorithms using MapReduce. In *2009 Ninth International Conference on Intelligent Systems Design and Applications*, pages 13–18. IEEE, 2009.

[218] A. F. Villaverde and J. R. Banga. Reverse engineering and identification in systems biology: strategies, perspectives and challenges. *Journal of the Royal Society Interface*, 11(91):20130505, 2014.

[219] A. F. Villaverde, J. A. Egea, and J. R. Banga. A cooperative strategy for parameter estimation in large scale systems biology models. *BMC Systems Biology*, 6(1):75, 2012.

[220] A. F. Villaverde, D. Henriques, K. Smallbone, S. Bongard, J. Schmid, D. Cicin-Sain, A. Crombach, J. Saez-Rodriguez, K. Mauch, E. Balsa-Canto, P. Mendes, J. Jaeger, and J. R. Banga. Biopredyn-bench: a suite of benchmark problems for dynamic modelling in systems biology. *BMC Systems Biology*, 9(1):1–15, 2015.

[221] T. D. Vo, W. P. Lee, and B. O. Palsson. Systems analysis of energy metabolism elucidates the affected respiratory chain complex in leigh's syndrome. *Molecular Genetics and Metabolism*, 91(1):15–22, 2007.

[222] H. Wang, S. Rahnamayan, and Z. Wu. Parallel differential evolution with self-adapting control parameters and generalized opposition-based learning for solving high-dimensional optimization problems. *Journal of Parallel and Distributed Computing*, 73(1):62 – 73, 2013.

[223] R.-S. Wang, A. Saadatpour, and R. Albert. Boolean modeling in systems biology: an overview of methodology and applications. *Physical biology*, 9(5):055001, 2012.

[224] R.-S. Wang, Y. Wang, X.-S. Zhang, and L. Chen. Inferring transcriptional regulatory networks from high-throughput data. *Bioinformatics*, 23(22):3056–3064, 2007.

[225] Y. Wang, T. Joshi, X.-S. Zhang, D. Xu, and L. Chen. Inferring gene regulatory networks from multiple microarray datasets. *Bioinformatics*, 22(19):2413–2420, 2006.

[226] M. Weber, F. Neri, and V. Tirronen. Distributed differential evolution with explorative–exploitative population families. *Genetic Programming and Evolvable Machines*, 10(4):343–371, 2009.

[227] M. Weber, F. Neri, and V. Tirronen. Shuffle or update parallel differential evolution for large-scale optimization. *Soft Computing*, 15(11):2089–2107, 2011.

[228] M. Weber, F. Neri, and V. Tirronen. A study on scale factor in distributed differential evolution. *Information Sciences*, 181(12):2488–2511, 2011.

[229] M. Weber, F. Neri, and V. Tirronen. A study on scale factor/crossover interaction in distributed differential evolution. *Artificial Intelligence Review*, 39(3):195–224, 2013.

[230] M. Weber, V. Tirronen, and F. Neri. Scale factor inheritance mechanism in distributed differential evolution. *Soft Computing*, 14(11):1187–1207, 2010.

[231] L. Weihmann, D. Martins, and L. dos Santos Coelho. Modified differential evolution approach for optimization of planar parallel manipulators force capabilities. *Expert Systems with Applications*, 39(6):6150 – 6156, 2012.

[232] D. M. Wittmann, J. Krumsiek, J. Saez-Rodriguez, D. A. Lauffenburger, S. Klamt, and F. J. Theis. Transforming boolean models to continuous models: methodology and application to t-cell receptor signaling. *BMC Systems Biology*, 3(1):98, 2009.

[233] O. Wolkenhauer and M. Mesarović. Feedback dynamics and cell function: Why systems biology is called systems biology. *Molecular BioSystems*, 1(1):14–16, 2005.

[234] X10 Parallel Programming Language. `http://x10-lang.org`.

[235] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[236] D. Zaharie and D. Petcu. Parallel implementation of multi-population differential evolution. In *Proceedings of the NATO Advanced Research Workshop on Concurrent Information Processing and Computing*, pages 223–232, 2003.

[237] Y. Zhai, M. Liu, J. Zhai, X. Ma, and W. Chen. Cloud versus in-house cluster: evaluating amazon cluster compute instances for running mpi applications. In *SC'11: State of the Practice Reports*, pages 11:1–11:10. ACM, 2011.

[238] S.-Z. Zhao, P. N. Suganthan, and S. Das. Self-adaptive differential evolution with multi-trajectory search for large-scale optimization. *Soft Computing*, 15(11):2175–2185, 2011.

[239] C. Zhou. Fast parallelization of differential evolution algorithm using MapReduce. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, pages 1113–1114. ACM, 2010.

[240] W. Zhu. Massively parallel differential evolution-pattern search optimization with graphics hardware acceleration: An investigation on bound constrained optimization problems. *Journal of Global Optimization*, 50(3):417–437, 2011.

[241] E. C. T. Zúñiga, I. L. L. Cruz, and A. R. García. Parameter estimation for crop growth model using evolutionary and bio-inspired algorithms. *Applied Soft Computing Journal*, 23(0):474–482, 2014.

Appendix A

Released codes

The implementations of the proposed algorithms in this Thesis have been released in the form of several publicly available tools. This appendix describes the main functionalities of these tools.

A.1. asynPDE library

The asynPDE library aims to solve NLP problems applying global optimization, parallelization techniques, and local searches. Version 0.1 of this tool includes the implementation of the island-based MPI asynchronous enhanced Differential Evolution algorithm presented in Chapter 2. This code incorporates several key new mechanisms:

- Asynchronous cooperation between parallel processes based on the island-model.
- Three enhanced strategies: the use of logarithmic space, a local solver and a tabu list, to improve the search.

Besides, for comparative purposes, this library also includes a synchronous implementation of the proposed method.

The local solver included in this library is the "Nonlinear Least-Squares Algorithm" (nl2sol) [54]. The version used is the nl2sol solver from the PORT library [170].

This version of the library also includes different benchmarks to be tested: BBOB [87] and a set of problems within computational systems biology:

- *Circadian* model: a parameter estimation problem in a dynamic model of the circadian clock in the plant *Arabidopsis thaliana*, as presented in [127].
- *NF-κB* model: a problem based on the two-feedback-loop regulatory module of nuclear factor kappa B (NF-κB) signaling pathway presented in [126].
- *3-step pathway model*: a problem considering a 3-step generic and highly nonlinear pathway, as presented in [141].

The asynPDE code has been implemented using C. MPI has been used in the parallel implementation. It has been tested in Linux clusters running CentOS 6.7 and Debian 8.

The source code of the asynPDE library is available at:

- `https://bitbucket.org/DavidPenas/asynpde`

A.2. saCeSS library

The saCeSS code is a tool distributed as a library with several parallel solvers based on the Scatter Search (SS) metaheuristic. The saCeSS library aims to solve NLP and MINLP problems. It also provides efficient local solvers for nonlinear parameter estimation problems associated with complex models. The current distribution of saCeSS includes a set of optimization examples that can be used as benchmarks, taken from the BBOB [87] and BioPreDyn [220] testbeds.

The library includes a set of different versions of SS:

- eSS: sequential version of enhanced Scatter Search [61].

- eSSm: eSS with multiple configurations, where a specific number of instances of sequential SS are executed in parallel without cooperation among them. This option corresponds to *np*-eSS algorithm described and used in Chapter 3 and Chapter 4.
- CeSS: Cooperative enhanced Scatter Search, i.e. a parallel cooperative eSS scheme where different eSS processes run in parallel and exchange information in a synchronous fashion (based on a pre-defined time interval), as described in [219].
- aCeSS: asynchronous Cooperative enhanced Scatter Search, a distributed asynchronous version as described in [166].
- saCeSS: Self-adaptive asynchronous Cooperative enhanced Scatter Search. This version corresponds with the method proposed in Chapter 3.

Local solvers are a key element in Scatter Search, significantly accelerating its convergence. Although Scatter Search can be executed without a local solver, this is only recommended for extremely pathological cost functions (e.g. very noisy, or with many sharp discontinuities). The local solvers provided with the current version of saCeSS, are:

- *nl2sol*, which stands for ”Nonlinear Least-Squares Algorithm” [54]. The version used is the nl2sol solver from the PORT library [170].
- *DHC*, which stands for ”Dynamic Hill Climbing” [52].

Figure A.1 summarizes the general structure of the saCeSS optimization library. The code is organized in several folders:

- benchmark: contains the source code of the different optimization problems integrated in this distribution. New problems may be add, modifying the template files in the *customized* folder.
- doc: documentation generated by Doxygen, in several formats (including HTML).

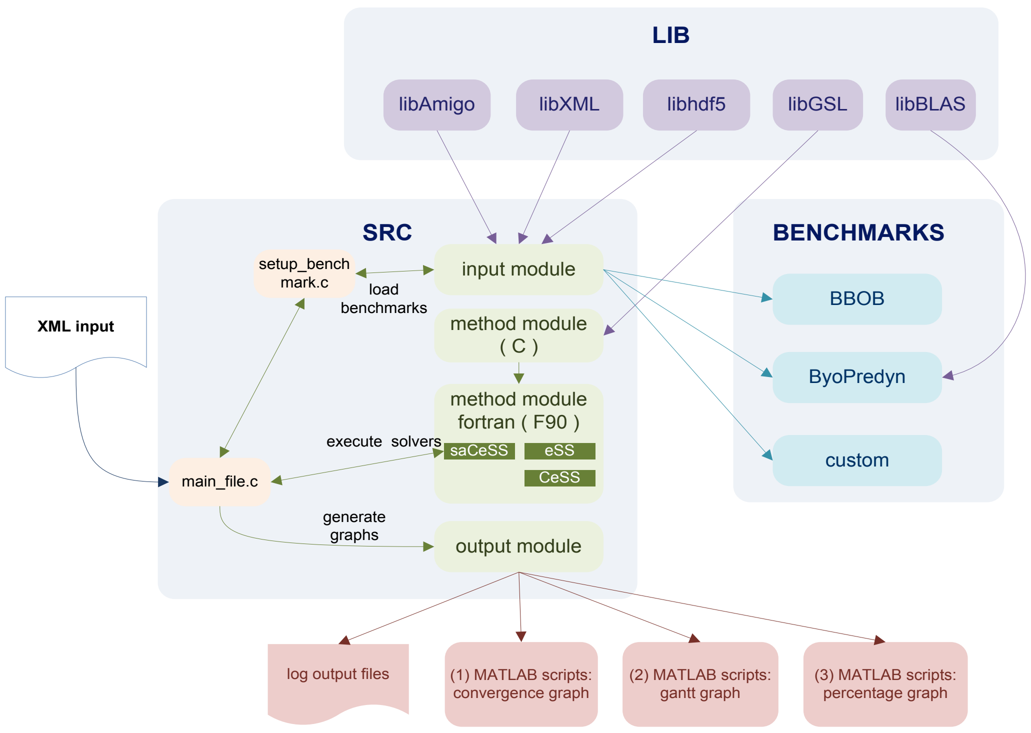


Figure A.1: Structure of saCeSS optimization library.

- include: header files of C functions.
- src: Fortran 90 and C files, with I/O modules.
- inputs: set of templates and examples for the required XML input file.
- lib: set of required libraries, such as BLAS, hdf5, libxml2, GSL, etc.
- output: folder for the results files generated in each execution.

The execution of these solvers produces the following output files:

- a main output file, reporting general information, such as optimization problem characteristics, search options enabled, final solution achieved, final computation time, etc.
- a specific log file per process.

- output plots: convergence plots and, in the case of saCeSS method, Gantt charts illustrating cooperation among processors, and plots detailing the search improvements history.

The saCeSS library has been implemented using Fortran 90 and C. Parallelization has been achieved through MPI and openMP. Current version of saCeSS library has the following requirements:

- Linux command line systems. The saCeSS library has been tested under CentOS 6.6, CentOS 6.7 and Debian 8.
- GNU or Intel compilers. The library has been compiled with GCC compiler (gcc-4.4.7) and Intel compilers (icc-13.1.1 and intel-14.0.2).
- openMPI or IntelMPI implementations of MPI. The saCeSS library has been compiled with openmpi-1.8.3 and openmpi-1.8.4. It has also been tested with intelmpi-5.0.

The source code of the saCeSS library is available at:

- `https://bitbucket.org/DavidPenas/sacess-library`

A.3. saCeSS2 library

The saCeSS2 library is a new version of the saCeSS library intended for the solution of large-scale mixed-integer dynamic optimization (MIDO) problems. This library incorporates these novel key features:

- Improved self-tuning strategies, suitable for large mixed-integer nonlinear optimization.
- An efficient mixed-integer nonlinear optimization local solver: *MISQP*, which stands for ”Mixed Integer Sequential Quadratic Programming” [69].

The set of very challenging case studies from the domain of computational biology (reverse engineering of cell signaling) used in Chapter 4 are provided with this distribution:

- SSP: the simpler case study, considering a synthetic signaling pathway with 84 continuous and 34 binary decision
- HePG2: a second case study considering the dynamic modeling of liver cancer data, with 135 continuous and 109 binaries decision variables.
- HPN-DREAM: an extremely difficult case study related with breast cancer, involving 690 continuous and 138 binary decision variables.

The saCeSS2 library has the following requirements:

- Linux command line systems. The saCeSS2 version has been tested under CentOS 6.6, CentOS 6.7 and Debian 8.
- GNU or Intel compilers. This version has been compiled with gcc-4.4.7 and with Intel compilers: intel-13.1.1 and intel-14.0.2.
- openMPI or IntelMPI implementation of MPI. The saCeSS2 version has been compiled with openmpi-1.8.3, openmpi-1.8.4 and intelmpi-5.0.
- R programming language should be installed in the system. This version has beed tested with R-3.2.0.

The source code of the saCeSS2 library is available at:

- `https://doi.org/10.5281/zenodo.290219`

Resumen en español

Muchos problemas de gran transcendencia en campos como la biología de sistemas o la bioinformática se pueden formular y resolver usando técnicas de optimización global. La complejidad existente en los modelos matemáticos desarrollados en estos campos requiere el uso de herramientas eficientes para obtener resultados satisfactorios en tiempos de cálculo razonables. En este contexto, las metaheurísticas son algoritmos de optimización estocásticos muy populares, con los que se puede localizar la vecindad de la solución global sin tener que explorar todo el espacio de direcciones disponible para un problema dado. De esta forma se reduce el número de evaluaciones y, como consecuencia, también el tiempo de ejecución. Sin embargo, cuando se trata de resolver problemas vinculados a muchas aplicaciones reales, es habitual que estos algoritmos todavía requieran un gran esfuerzo computacional y demasiado tiempo de ejecución. Esta tesis se centra en desarrollar nuevas metaheurísticas que permitan acelerar la resolución de muchos de estos problemas, planteados como problemas de programación no lineal (NLP) o bien como problemas de programación no lineal entera mixta (MINLP). Ambos tipos de problemas están sujetos a restricciones dinámicas no lineales de igualdad o desigualdad, que convierten la tarea de optimización en algo muy complejo debido a la multimodalidad y a la naturaleza no convexa de dichos procesos.

En la actualidad existe una gran cantidad de infraestructuras multiprocesador fácilmente accesibles, como pequeños clústeres computacionales existentes en multitud de departamentos y centros de investigación, ciertos superordenadores en grandes centros de cálculo, o también recientes instalaciones de computación en la nube. Usando estas infraestructuras se puede mejorar el rendimiento de las metaheurísticas para resolver costosos problemas utilizando técnicas de computación de altas prestaciones (HPC). La paralelización de las metaheurísticas persigue uno o más de los siguientes objetivos: aumentar el tamaño de los problemas que se pueden resolver, acelerar los cálculos efectuados, y/o intentar una exploración más completa del espacio de la solución. Sin embargo, lograr una paralelización eficiente de estos algoritmos no es una tarea sencilla, ya que, típicamente en estos algoritmos, la búsqueda de nuevas soluciones depende de las iteraciones previas, lo que no solo complica la paralelización sino que también limita la escalabilidad.

Esta tesis propone el diseño, implementación y evaluación de diferentes metaheurísticas paralelas distribuidas, aplicadas a problemas NLP o MINLP en procesos de optimización extremadamente difíciles dentro del campo de la biología de sistemas. Los algoritmos propuestos en los sucesivos capítulos de esta tesis se basan en reconocidas metaheurísticas, como el algoritmo de evolución diferencial (*Differential Evolution* - DE) y el algoritmo de búsqueda dispersa (*Scatter Search* - SS). También se hace una incursión en la computación en la nube, evaluando las propuestas en una infraestructura pública, Microsoft Azure.

La tesis se organiza en cinco capítulos. El capítulo 1 contextualiza de forma breve el tipo de problemas que esta tesis pretende resolver, y los medios que se han usado para alcanzar este objetivo. El capítulo 2 presenta el diseño e implementación de una versión paralela asíncrona del algoritmo DE. El capítulo 3 describe una nueva metaheurística paralela basada en el método de búsqueda dispersa mejorada (*enhanced Scatter Search* - eSS). El capítulo 4 se centra en la modificación de la propuesta anterior para su uso en problemas MINLP. Y, por último, el capítulo 5 explora el rendimiento de las propuestas anteriores en una infraestructura en la nube, y lo compara con los resultados obtenidos en clústeres locales y supercomputadores. En las siguientes secciones se resumen las principales ideas de cada capítulo.

Evolución diferencial mejorada paralela

El algoritmo de evolución diferencial (DE) [194] es una de las heurísticas más populares en optimización global, y se ha utilizado con éxito en áreas muy diferentes [42,48,172]. En la actualidad, el DE sigue siendo un método ampliamente usado para la identificación paramétrica de modelos complejos [47,175,241]. Sin embargo, en la mayoría de aplicaciones reales, este método poblacional requiere un gran número de evaluaciones de la función de coste, y en consecuencia mucho tiempo, para obtener un resultado aceptable.

En el capítulo 2 de esta tesis se presenta una mejora del algoritmo DE diseñado para resolver problemas muy complejos en el campo de la biología de sistemas. El método propuesto, llamado *evolución diferencial paralela asíncrona* (*asynchronous Parallel Differential Evolution* - asynPDE), tiene como objetivo conseguir un buen equilibrio entre la exploración del espacio de búsqueda y la intensificación de la misma. Primero, se propone una paralelización del algoritmo usando un protocolo asíncrono de comunicaciones que consigue no solo reducir el tiempo de computación, gracias al reparto de trabajo entre los recursos computacionales disponibles, sino también incrementar el espacio de búsqueda, a través de la exploración simultánea y concurrente en diferentes espacios. Segundo, se incluye un método de búsqueda local y otras mejoras algorítmicas, como una lista tabú para controlar las soluciones que entran en la mencionada búsqueda local, o la búsqueda en el espacio logarítmico. Así se mejora la propuesta clásica del algoritmo DE a través de la intensificación en la búsqueda, reduciendo drásticamente el número de evaluaciones necesarias para llegar a las proximidades del óptimo global.

La nueva metaheurística propuesta se ha probado utilizando diferentes tipos de problemas. Por un lado se ha usado una serie de programas de prueba populares en optimización, como es el conjunto de pruebas BBOB [87]. Por otro lado se han empleado tres problemas de estimación de parámetros en el campo de la biología de sistemas que son especialmente difíciles de resolver: el reloj circadiano de la planta *Arabidopsis thaliana* [127], el modelo *NF-κB* descrito en [126], y un modelo de camino en tres pasos presentado en [141]. Cabe señalar que, aunque el método presentado aquí se basa en un método híbrido (global-local), las mejoras heurísticas introducidas son fundamentales para explotar con éxito las características especiales de los problemas de estimación de parámetros en biología de sistemas que usamos en nuestra evaluación, ya que están muy mal acondicionados y presentan una gran multimodalidad [218].

Como ejemplo de la importancia práctica de esta propuesta, destacamos los resultados obtenidos para uno de los problemas considerados en las pruebas, el modelo de camino en tres pasos. Típicamente este problema necesita más de 3 días de tiempo de cálculo si se utiliza una versión clásica del algoritmo DE en un único procesador. En cambio, usando el método propuesto, asynPDE, este problema se puede resolver en menos de un minuto con 10 procesadores.

Aunque el método propuesto fue diseñado e implementado teniendo en cuenta las características que presentan los problemas de estimación de parámetros en biología de sistemas, también puede aplicarse a la resolución de otro tipo de problemas de optimización global, como se ha demostrado en la evaluación experimental llevada a cabo.

Búsqueda dispersa mejorada cooperativa y auto-adaptativa

El algoritmo de búsqueda dispersa (SS) [81] es una metaheurística poblacional muy prometedora para resolver problemas de optimización combinatoria y no lineal. Este método evolutivo utiliza estrategias que combinan vectores solución almacenados en una pequeña población con el fin de obtener buenos resultados sin gastar muchos recursos. En la literatura, se han propuesto diferentes implementaciones en las que SS demuestra que puede superar a otros métodos de optimización global estocásticos de última generación [22, 50, 61, 71, 74, 94, 104, 181]. Sin embargo, de nuevo cuando se trata de aplicar esta metaheurística a problemas de optimización vinculados a modelos de casos reales, la complejidad asociada es tan grande que el algoritmo SS necesita un tiempo de cálculo inasumible para obtener soluciones de calidad.

En el capítulo 3 se propone una versión paralela y mejorada del algoritmo SS llamado: *búsqueda dispersa mejorada cooperativa y auto-adaptativa* (*self-adaptive Cooperative enhanced Scatter Search* - saCeSS). Esta nueva metaheurística propone una estrategia paralela que incorpora:

- un esquema cooperativo, que incluye un mecanismo de intercambio de soluciones basado en la calidad de estas.
- un protocolo de comunicación asíncrono para gestionar el intercambio de información entre procesos.

- la combinación de una solución paralela de grano grueso usando MPI y una paralelización de grano fino con openMP, con el fin de mejorar la escalabilidad de la propuesta final.
- un procedimiento autoadaptativo para ajustar dinámicamente los parámetros de configuración de las búsquedas paralelas durante el progreso del algoritmo.

El rendimiento y la escalabilidad de este nuevo método se ilustra usando un conjunto de problemas muy complejos de estimación de parámetros en modelos dinámicos de gran escala en el campo de la biología de sistemas. Estos problemas abarcan una variedad de modelos cinéticos, como el de la bacteria *E. Coli*, la levadura de panadería *S. Cerevisiae*, la mosca del vinagre *D. Melanogaster*, las células de ovario de hámster chino, y una red genérica de transducción de señales.

Los resultados obtenidos y el análisis estadístico realizado para comparar la solución propuesta con otras versiones del algoritmo SS, muestran que la estrategia cooperativa y autoadaptativa propuesta (saCeSS) es robusta y eficiente, permitiendo una reducción muy significativa del tiempo de cálculo con respecto a otros métodos, incluso cuando se utiliza un pequeño número de procesadores. Por ejemplo, el modelo de las células de ovario de hámster chino necesita más de 10 horas usando el método de búsqueda dispersa mejorada (eSS) secuencial, y puede ser resuelto en cinco minutos usando saCeSS con 10 procesadores. Los resultados experimentales demuestran que el método propuesto puede desempeñar un papel clave en el desarrollo de modelos dinámicos a gran escala en biología de sistemas.

Búsqueda dispersa en problemas de optimización dinámica entera mixta

Durante la última década ha crecido de forma significativa el interés en modelar la dinámica de los sistemas biológicos. Como consecuencia, se están realizando importantes esfuerzos en la explotación de estos modelos dinámicos mediante técnicas de optimización matemática. En este contexto, el capítulo 4 de esta tesis se centra en la resolución de problemas de optimización dinámica entera mixta (MIDO), donde parte de las variables de decisión son discretas (binarias o enteras) [41].

Muchos problemas de optimización dinámica se centran en extraer políticas y/o diseños operativos útiles de un modelo dinámico. Dichas formulaciones también pueden aplicarse al propio proceso de construcción del modelo, es decir, al llamado problema de ingeniería inversa [56,75,88,103,113,116,138,188], que es extremadamente difícil [218]. Dado que el método saCeSS descrito en el capítulo 3 ha demostrado su potencial para resolver problemas NLP muy complejos, en el capítulo 4 se presenta una nueva versión, llamada saCeSS2, en la que el algoritmo saCeSS se modifica para manejar y resolver problemas MIDO y MINLP. Para ello, se han incluido las siguientes características en la nueva implementación propuesta:

- un método eficiente de búsqueda local especializado en enteros mixtos (MISQP)
- un nuevo mecanismo de autoadaptación para evitar el estancamiento de la convergencia
- la inyección de diversidad extra durante las tareas de adaptación, reiniciando la mayor parte de las poblaciones de los procesos reconfigurados

Los resultados obtenidos al resolver una serie de casos de estudio muy complejos en biología de sistemas, planteados como problemas MIDO-MINLP, muestran que la propuesta reduce significativamente el tiempo de ejecución necesario para obtener una solución de calidad. Por ejemplo, en el tercer y más complejo caso de estudio, se puede alcanzar un valor de la función objetivo de 41000 en poco más de 24 horas usando saCeSS2 con 40 procesadores, frente a los más de 10 días que se necesitan usando la versión eSS secuencial. Además, la dispersión en los resultados es menor a medida que aumenta el número de procesadores utilizados. Estos resultados confirman que el método puede utilizarse para realizar ingeniería inversa en modelos dinámicos muy complejos, y también podría ser adecuado para otras aplicaciones basadas en la optimización a gran escala de enteros mixtos, como la ingeniería metabólica [192], la planificación óptima de medicamentos [44, 57], o en biología sintética [158].

Computación de metaheurísticas paralelas en la nube

La computación en la nube ha surgido en los últimos años como un nuevo paradigma donde los recursos de computación se ofrecen al usuario bajo demanda. Con la aparición de esta nueva tecnología, el acceso a un gran número de recursos distribuidos resulta mucho más sencillo. Sin embargo, diversas razones han frenado su adopción por parte de la comunidad HPC. Primero, debido a la dificultad de emplear los recursos de computación en la nube. La curva de aprendizaje para entender las diferentes arquitecturas y entornos de ejecución, y también las diversas plataformas existentes, complica la adopción como alternativa de cómputo por parte de la comunidad HPC. Y segundo, porque los recursos en la nube todavía presentan desafíos importantes en aspectos como su prestación computacional. Recientemente algunos trabajos de investigación se han centrado en evaluar el uso de estas plataformas con aplicaciones HPC [63, 66, 70, 102, 147], y concluyen que, en la mayoría de los casos, estas infraestructuras necesitan una mejora significativa en cuanto a prestaciones para convertirse en una alternativa competitiva a los recursos HPC tradicionales.

En el capítulo 5 se presenta la evaluación en Microsoft Azure, una infraestructura de nube pública, de los diferentes algoritmos paralelos propuestos en esta tesis. Primero, se ha explorado y comparado el rendimiento de problemas de estimación de parámetros en biología de sistemas, usando por un lado la implementación asíncrona paralela del algoritmo DE, asynPDE, propuesta en el capítulo 2, que usa MPI (orientado a la computación de alto rendimiento), y por otro lado una implementación paralela del algoritmo DE basada en Spark (orientado hacia la productividad). Los resultados obtenidos en Azure se han comparado con aquellos obtenidos en clústeres locales. Los resultados muestran que, como se esperaba, desde un punto de vista computacional la implementación MPI supera a la realizada con Spark en términos de tiempo de ejecución. Sin embargo, existen aspectos positivos en la versión desarrollada en Spark que merece la pena tener en cuenta, como que su programación es más sencilla o que incluye soporte de tolerancia a fallos y replicación de datos.

A mayores, se ha evaluado el rendimiento del método de búsqueda dispersa mejorada paralelo para MIDO, saCeSS2, propuesto en el capítulo 4, también en Microsoft Azure. Se ha observado que el comportamiento del algoritmo es ligeramente diferente en el clúster local que en la plataforma Azure. En particular, los resultados para un número relativamente pequeño de procesadores son mejores en Azure que en el clúster local. Sin embargo, los resultados en el clúster superan en rendimiento a los obtenidos en Azure cuando el número de procesadores crece. Este comportamiento se atribuye a la eficiencia de las comunicaciones intra-nodo cuando usamos las instancias Azure para cómputo intensivo, frente a la mayor latencia en las comunicaciones inter-nodo.

Conclusiones y trabajo futuro

Las técnicas de optimización global se utilizan cada vez más en diversos campos de la ingeniería y en la mayoría de las ciencias básicas y aplicadas, como la bioinformática o la biología de sistemas. En el caso de procesos químicos y biológicos, durante la última década ha sido creciente el interés por modelar su dinámica, es decir, por desarrollar modelos cinéticos que sean capaces de encapsular la naturaleza variable en el tiempo de estos sistemas. Como consecuencia, muchas investigaciones se están centrando en explotar estos modelos dinámicos mediante técnicas de optimización matemática. Las metaheurísticas están ganando reconocimiento en este contexto, sin embargo, para las aplicaciones más realistas todavía se necesitan tiempos de cálculo excesivos. Con el fin de reducir este coste temporal, esta tesis ha realizado las siguientes contribuciones:

- Se ha diseñado un algoritmo paralelo de evolución diferencial (DE), llamado asynPDE, con el objetivo de resolver problemas complejos dentro de la biología de sistemas. La idea clave detrás de esta propuesta ha sido lograr un equilibrio adecuado de las habilidades de exploración existentes en el algoritmo DE y en las capacidades de explotación existentes en los algoritmos de búsqueda local. Así, el método consigue mejorar la búsqueda global mediante una implementación paralela asíncrona basada en un modelo de isla cooperativo, e incluyendo también un método de búsqueda local mejorada junto con la implementación de varias heurísticas adicionales. Los resultados experimentales muestran que (i) el tiempo de convergencia se reduce en varios órdenes de magnitud cuando se introduce un método local en el algoritmo DE, y (ii) la estrategia paralela asíncrona propuesta consigue una reducción adicional del tiempo de convergencia, demostrando también una aceleración importante frente a otras propuestas con comunicaciones síncronas.

- Se ha presentado un nuevo método distribuído basado en el algoritmo de búsqueda dispersa (SS), denominado algoritmo de búsqueda dispersa mejorada cooperativa autoadaptable (saCeSS). Las principales características de esta nueva implementación han sido: (i) una paralelización de grano grueso utilizando un esquema maestro-esclavo centralizado, enfocada a estimular la diversificación en la búsqueda y la cooperación entre diferentes procesos; (ii) una paralelización a grano fino para realizar evaluaciones de la función de coste concurrentemente y, por lo tanto, acelerar la búsqueda global; (iii) una cooperación entre procesos impulsada por la calidad de las soluciones encontradas; (iv) un protocolo de comunicación asíncrono para evitar períodos de espera entre los procesos; y por último (v) un mecanismo dinámico de autoadaptación de los parámetros de configuración de cada proceso distribuído, que se realiza basándose en el comportamiento de cada uno de ellos a lo largo de la ejecución. En su evaluación se han considerado problemas complejos de estimación de parámetros de modelos dinámicos a gran escala de sistemas biológicos. El método saCeSS ha sido comparado tanto con versiones secuenciales del algoritmo SS, como con otras implementaciones paralelas de este algoritmo, obteniendo una mejora substancial frente a todas ellas, tanto en rendimiento como en escalabilidad. Los resultados muestran que saCeSS es un método robusto y eficiente, que permite una reducción muy significativa de los tiempos de cálculo con respecto a los métodos anteriores del estado del arte.
- Se ha desarrollado una extensión del método saCeSS, denominada saCeSS2, para resolver con éxito problemas complejos de optimización dinámica entera mixta (MIDO). Esta extensión ha consistido por una parte en la inclusión de un método local adecuado para manejar problemas MIDO, y por otra parte en introducir cambios en el algoritmo que eviten el habitual estancamiento a la hora de resolver este tipo de problemas: el mecanismo que dispara la reconfiguración se ha modificado, y a la vez se inyecta una mayor diversidad durante las etapas de adaptación. En la evaluación de la propuesta se ha aplicado saCeSS2 a un conjunto de problemas MIDO-MINLP en los que resulta

muy difícil alcanzar buenas soluciones en un tiempo razonable. Los resultados obtenidos muestran una importante reducción en los tiempos de cálculo.

- Se ha realizado una evaluación de los diferentes métodos propuestos en esta tesis en una infraestructura de nube pública, en concreto en Microsoft Azure. Por un lado, la implementación asynPDE que se ha propuesto, orientada a la computación de altas prestaciones, se ha comparado con una implementación basada en Spark, orientada a la productividad. Se ha proporcionado un análisis detallado de las diferencias que surgen en ambos modelos, fruto de las características inherentes al modelo de programación que se ha utilizado en cada implementación, apoyado por la evaluación de los resultados experimentales. Además, también se ha evaluado el rendimiento del método saCeSS2 aplicado a los problemas MIDO-MINLP en Microsoft Azure.

Como se ha comentado ampliamente a lo largo de esta tesis, las metaheurísticas se han posicionado como una alternativa exitosa para solucionar una gran variedad de problemas de optimización. Desafortunadamente, no siempre es fácil, o incluso posible, anticipar cuál de los numerosos algoritmos ya existentes será el más adecuado para resolver un problema particular. Esta incertidumbre no sólo se limita a diferentes métodos en diferentes clases de problemas. Puede haber incluso problemas que sufran grandes variaciones en el rendimiento del algoritmo entre diferentes instancias del mismo. Por lo tanto, en relación con el trabajo futuro, nuestra investigación se centrará en ampliar los métodos desarrollados y las implementaciones para ser útiles a una gama más amplia de aplicaciones. En particular, nos centraremos en:

- extender los métodos saCeSS y saCeSS2 para manejar problemas de optimización multiobjetivo, ya que muchos problemas que son clave en la biología de sistemas implican la optimización simultánea de múltiples objetivos que entran en conflicto.
- incorporar métodos locales adicionales, especialmente para el caso de problemas MINLP.
- generalizar la idea de búsqueda cooperativa autoadaptativa, explorada en esta tesis, a través del concepto de optimización *multimétodo*, en la que múltiples

algoritmos de búsqueda se podrían ejecutar concurrentemente y cooperando entre ellos a través del intercambio de información.

- considerar la optimización en aplicaciones que constituyen un desafío especial en el diseño a gran escala en biología sintética.

Finalmente, aunque esta investigación se ha centrado en buscar soluciones eficientes en el campo de la estimación de parámetros en biología de sistemas, creemos que los resultados obtenidos en esta tesis pueden ser útiles para aquellos investigadores interesados en el rendimiento de metaheurísticas paralelas en optimización global, bien sea en clústeres locales, supercomputadores o nuevas plataformas de computación en la nube, así como también a aquellos interesados en el potencial de los modelos de programación paralelos para desarrollar nuevas metaheurísticas paralelas.

Todas las propuestas realizadas en esta tesis, incluyendo tanto el código desarrollado como los ficheros de datos necesarios para reproducir los resultados mostrados aquí, se encuentran publicamente disponibles en:

- asynPDE:
  `https://bitbucket.org/DavidPenas/asynpde`
- saCeSS:
  `https://bitbucket.org/DavidPenas/sacess-library`
- saCeSS2:
  `https://doi.org/10.5281/zenodo.290219`

Los resultados de los trabajos de investigación que constituyen esta tesis han sido publicados (o están en proceso de revisión) en las siguiente revistas y conferencias:

- Revistas internacionales (4):
  - D. R. Penas, J. R. Banga, P. González, y R. Doallo. Enhanced parallel differential evolution algorithm for problems in computational systems biology. *Applied Soft Computing*, 33:86–99, 2015. [165]

    Indexada en JCR. Factor de impacto (JCR 2015): 2.857. Ranking: Q1 [16/104].

- D. R. Penas, P. González, J. A. Egea, R. Doallo, y J. R. Banga. Parameter estimation in large-scale systems biology models: a parallel and self-adaptive cooperative strategy. *BMC Bioinformatics*, 18(1):52, 2017. [167]

  Indexada en JCR. Factor de impacto (JCR 2015): 2.435. Ranking: Q1 [10/56].

- D. Teijeiro, X. C. Pardo, D. R. Penas, P. González, J. R. Banga, y R. Doallo. A cloud-based enhanced differential evolution algorithm for parameter estimation problems in computational systems biology. *Cluster Computing*, en revisión menor. [201]

  Indexada en JCR. Factor de impacto (JCR 2015): 1.514. Ranking: Q2 [28/105].

- D. R. Penas, D. Henriques, P. González, R. Doallo, J. Saez-Rodriguez, y J. R. Banga. A parallel metaheuristic for large mixed-integer nonlinear dynamic optimization problems, with applications in computational biology. *PLOS ONE*, en revisión. [168]

  Indexada en JCR. Factor de impacto (JCR 2015): 3.057. Ranking: Q1 [11/63].

▪ Conferencias internacionales (6):

- D. R. Penas, J. R. Banga, P. González, y R. Doallo. A parallel differential evolution algorithm for parameter estimation in dynamic models of biological systems. *Advances in Intelligent Systems and Computing*, 294:173–181, 2014. Proceedings of the 8th International Conference on Practical Applications of Computational Biology & Bioinformatics (PACBB 2014). [164]

- D. R. Penas, P. González, J. A. Egea, J. R. Banga, y R. Doallo. Parallel metaheuristics in computational biology: an asynchronous cooperative enhanced scatter search method. *Procedia Computer Science*, 51:630 – 639, 2015. Proceedings of the International Conference On Computational Science (ICCS 2015). [166]

- D. R. Penas, P. González, R. Doallo, J. A. Egea y J. R. Banga. An asynchronous cooperative search metaheuristic for computational systems biology problems. *Advanced Lecture Course on Computational Systems Biology* (CompSysBio 2015). Presentación poster.

- D. R. Penas, P. González, D. Henriques, J. Saez-Rodriguez, R. Doallo, J. R. Banga. A parallel global optimization method to reverse engineer dynamic models of complex biochemical pathways. *Bioinformatics for Young international researchers Expo: Maastricht-Aachen-Liège* (byte-MAL 2016). Presentación poster.
- D. R. Penas, P. González, J. A. Egea, R. Doallo, J. R. Banga. Development of large-scale dynamic models of complex biochemical pathways via high-performance computational optimization. *17th International Conference on Systems Biology* (ICSB 2016). Presentación poster.
- P. González, X. C. Pardo, D. R. Penas, D. Teijeiro, J. R. Banga, y R. Doallo. Using the Cloud for parameter estimation problems: comparing Spark vs MPI with a case-study. *Workshop on Clusters, Clouds and Grids for Life Sciences*, in conjunction with CCGrid 2017 Conference. [82]