UNIVERSIDAD DE EXTREMADURA

TESIS DOCTORAL

Título: "Análisis e Inferencia Multiobjetivo de Hipótesis Filogenéticas mediante Computación Paralela y Bioinspirada"

Title: "Multiobjective Analysis and Inference of Phylogenetic Hypotheses by Means of Parallel and Bioinspired Computing"

Autor: Sergio Santander Jiménez

Departamento: Tecnología de los Computadores y de las Comunicaciones

Conformidad del director:

Fdo: Dr. Miguel Ángel Vega Rodríguez

Año de lectura: 2015

Multiobjective Analysis and Inference of Phylogenetic Hypotheses by Means of Parallel and Bioinspired Computing

Análisis e Inferencia Multiobjetivo de Hipótesis Filogenéticas mediante Computación Paralela y Bioinspirada

Author: Sergio Santander-Jiménez
Advisor: Miguel A. Vega-Rodríguez, Ph.D

Department of Computer and Communications Technologies

Escuela Politécnica - University of Extremadura, Cáceres (Spain)

This dissertation is submitted for the degree of

Doctor of Philosophy in Computer Science

October 2015

To my beloved family . . .

Abstract

One of the most outstanding research topics in the field of bioinformatics is the reconstruction of evolutionary relationships among species. By studying the molecular features of living organisms, phylogenetic inference procedures seek to provide hypotheses about the evolutionary events which led to the current biodiversity in nature. The result of a phylogenetic analysis is given by a tree-shaped data structure known as phylogenetic tree, whose topology describes the evolutionary history of the input species by defining ancestor-descendant relationships. The biological quality of the inferred phylogenies is usually evaluated by means of optimality criteria, which allow the modelling of phylogenetic inference as an optimization problem.

In this context, several key problems must be addressed. Firstly, the reconstruction of optimal solutions implies the processing of a search space whose size grows exponentially with the number of species under study. Additional difficulties are given by the fact that evaluation procedures require complex computations whose number grows linearly with the length of the input molecular sequences. A more controversial problem lies on the choice of the preferred optimality criterion, as it represents one of the most troublesome sources of conflict in phylogenetics. Situations where different optimality criteria give support to conflicting evolutionary histories for a given dataset can be solved by proposing a compromise view of phylogenetics based on multiobjective optimization.

This PhD Thesis aims to apply bioinspired computing to perform real phylogenetic analyses according to the maximum parsimony and maximum likelihood criteria. Such goal requires a multiobjective formulation of the problem which considers the optimization of both objective functions simultaneously, in such a way that the output of the procedure will be given not by a single solution, but a set of multiobjective phylogenetic trees which represent good compromises between divergent criteria. The implications of performing multiobjective phylogenetic analyses have not been fully evaluated in the literature, so this research seeks to carry out comprehensive comparative studies among different multiobjective algorithms, in order to decide which algorithmic designs maximize the quality of the inferred trees from a multiobjective and biological point of view.

Due to the significant complexity of problem, this Thesis is especially focused on the exploitation of modern parallel architectures to conduct real phylogenetic analyses. More specifically, this research undertakes the introduction of parallel computing techniques to the applied multiobjective designs, with the aim of taking advantage of the computing capabilities offered by different hardware setups: shared-memory architectures, multicore clusters, and heterogeneous CPU+GPU systems.

In summary, this PhD Thesis is focused on the application of parallel and bioinspired computing to tackle the phylogenetic inference problem. The key goals of this research include the definition of a multiobjective formulation of phylogenetics to address conflicts in real biological analyses, the study and assessment of a variety of bioinspired multiobjective designs, and their efficient parallelization on different hardware architectures. Through the comparison with other state-of-the-art phylogenetic tools, we give account of the relevant parallel, multiobjective, and biological performance attained by the proposed multiobjective designs.

Resumen

La reconstrucción de relaciones evolutivas representa una de las cuestiones más significativas en bioinformática. Mediante el estudio de las caracteristicas moleculares de los organismos vivos, los métodos de inferencia filogenética pretenden proporcionar hipótesis sobre los eventos evolutivos que condujeron a la biodiversidad presente en la naturaleza. El resultado de un análisis filogenético viene dado por una estructura de datos denominada árbol filogenético, cuya topología describe la historia evolutiva de las especies de entrada mediante la definición de relaciones ancestro-descendiente. La calidad biológica de las filogenias obtenidas es evaluada mediante el uso de criterios de optimalidad, los cuales permiten el modelado del proceso de inferencia como un problema de optimización.

En este contexto, es necesario afrontar una serie de problemas clave. En primer lugar, la inferencia de soluciones óptimas conlleva el procesamiento de un espacio de búsqueda cuyo tamaño crece exponencialmente con el número de especies a estudiar. Además, los pasos de evaluación precisan realizar cálculos complejos que aumentan linealmente en número con la longitud de las secuencias moleculares de entrada. Una cuestión más sensible radica en la elección del criterio de optimalidad a seguir, la cual representa una de las fuentes de conflicto más problemáticas en filogenética. Aquellas situaciones en que distintos criterios dan soporte a historias evolutivas conflictivas para una misma base de datos biológica pueden ser resueltas mediante la propuesta de una visión de compromiso de la filogenética, basada en el empleo de técnicas de optimización multiobjetivo.

Esta Tesis Doctoral estudia la aplicación de computación bioinspirada para inferencia filogenética de acuerdo a los criterios de máxima parsimonia y máxima verosimilitud. Esto requiere proponer una formulación multiobjetivo del problema que considere la optimización simultánea de ambas funciones objetivo, de tal manera que la salida del procedimiento no vendrá dada por una única solución, sino por un conjunto de filogenias multiobjetivo que representen compromisos entre criterios divergentes. Las implicaciones de realizar análisis filogenéticos multiobjetivo no han sido evaluadas en profundidad en la literatura, por lo que esta investigación pretende aportar estudios comparativos entre distintos algoritmos multiobjetivo, con objeto de arrojar luz sobre qué diseño algorítmico maximiza la calidad de los árboles inferidos desde un plano multiobjetivo y biológico.

Dada la significativa complejidad del problema, esta Tesis está especialmente enfocada en la explotación de las modernas arquitecturas paralelas para acometer análisis filogenéticos reales. Concretamente, la investigación considera la introducción de técnicas de computación paralela sobre los diseños multiobjetivo aplicados, con la idea de aprovechar las capacidades de cómputo paralelo

presentes en distintas configuraciones hardware: arquitecturas de memoria compartida, clusters multicore, y sistemas heterogéneos CPU+GPU.

En resumen, esta Tesis Doctoral aborda el estudio de computación paralela y bioinspirada para afrontar el problema de la inferencia filogenética. Los objetivos fundamentales incluyen la definición de una formulación multiobjetivo de la inferencia filogenética para afrontar los conflictos que se dan en análisis biológicos reales, el estudio y evaluación de diversos diseños multiobjetivo y bioinspirados, y su eficiente paralelización en distintas arquitecturas hardware. A través de estudios comparativos con otras herramientas filogenéticas del estado del arte, damos cuenta del relevante rendimiento paralelo, multiobjetivo y biológico obtenido por los diseños multiobjetivo propuestos.

Acknowledgements

I am not a man of many compliments but certainly I would like to acknowledge the support of a number of important people without whom this PhD Thesis would not have been possible. First of all, I want to thank my parents and brothers for all their love and support in this stage of my life. I hope this Thesis represents a better future for all of us.

Secondly, I would like to sincerely thank my advisor, Miguel Ángel, who is probably the most hard-working person I have ever met. I have no words to describe how grateful I am to you. Thank you for all your help and advice throughout the development of this research. I also really appreciate your support and guidance during my first teaching experiences. Definitely, it was a pleasure for me to work under your direction.

Many thanks to all my workmates and also friends at the ARCO Lab: Álvaro, Chema, David, Víctor, and other former members. I really enjoyed all the good times we spent together, helping me to feel the laboratory like my second home. In addition, I would like to acknowledge the support received from all the people I met at INESC-ID and Residence Baldaques during my research stay in Lisbon. Special thanks to Leonel, Aleksandar, and Diego, who welcomed me at their lab and helped me to accomplish satisfactorily my goals during the stay.

Thanks to my friends Alberto, Álvaro, Diego, Diego Manuel, Javi, Paco, and Pepe. You are really the best. Thanks also to the people from the Spanish FGC, Antonio, Carlos, Dani, David, Guille, Jordi, Juanma, Luis, Miguel Andrés, Zeben, and others. I hope to see all of you again in future events.

Finally, I would like to acknowledge the economical support received from the following institutions:

- Spanish Ministry of Education, Culture and Sports, for the predoc research grant FPU12/04101 (*Programa de Formación del Profesorado Universitario, convocatoria 2012*).
- Spanish Ministry of Economy and Competitiveness and the ERDF (European Regional Development Fund), for the financial support granted to the ARCO research group under the contract TIN2012-30685 (BIO project, 2013-2015).

Table of contents

List of figures

List of tables

Chapter 1

Introduction

This chapter introduces the main issues which motivate the research undertaken in this PhD Thesis, describing its main goals and the steps followed throughout its development. It also reports the main contributions of the Thesis and defines the organization of this document.

1.1 Motivation of the Research

Research interest in solving computationally demanding problems from the biological domain has grown significantly in the last decade. Advances in molecular sequencing have led to the generation of increasing amounts of biological data whose processing represents a significant challenge from both biological and computational perspectives [156]. Furthermore, the bioinformatics area encloses a wide range of applications which are built upon the modelling of high-complexity biological processes. Such processes often involve optimization procedures which seek to find the most satisfying solution in accordance with an optimality criterion which assesses its biological relevance [132]. In the current context, the way of tackling such problems represents a significant question from a research perspective, as most of these biological problems show an NP-hard nature that makes infeasible the attainment of high-quality solutions under classic algorithmic techniques.

In addition, we must bear in mind that the behaviour of real natural systems is usually governed by a number of different biological principles. In this sense, the application of single-criterion optimization procedures is often not suitable to deal with these problems, as different optimality criteria can lead to conflicting solutions for a same biological dataset. This fact explains the recent interest on tackling biological problems by means of multiobjective optimization techniques [77]. A multiobjective optimizer seeks to find not a single solution to the problem, but a set of Pareto solutions which optimize simultaneously multiple objective functions [36]. This formulation brings about new issues due to the additional complexity required to satisfactorily solve a multiobjective optimization problem (MOP), mostly motivated by the high processing times demanded to find and evaluate solutions according to multiple optimality criteria.

In order to address these issues, it is required the development of new algorithmic approaches designed to maximize result quality while minimizing execution time. Computer engineering can give support to the biology domain by means of two useful paradigms: bioinspired computing and parallelism. By mimicking the way natural organisms deal with problems, bioinspired computing provides algorithmic approaches to obtain high-quality solutions without requiring the processing times shown by exhaustive optimization methods. Therefore, a bioinspired metaheuristic [176] is defined as a general-purpose resolution scheme that can be adapted to any type of optimization problem, applying nature-inspired strategies to conduct the search for optimal solutions. In addition, recent advances in parallel architectures and high performance computing also play a key role nowadays in the solution of time-consuming optimization procedures. The availability of shared-memory multicore multiprocessors, hybrid shared-distributed memory cluster platforms, and heterogeneous CPU+GPU systems has opened the door to new opportunities for the development of parallel solutions to tackle complex biological problems. By combining bioinspired search mechanisms and parallelism, we can provide biologists with new computational approaches to address representative questions in life, such as the reconstruction of the evolutionary history of living species [105].

Phylogenetic inference is considered one of the main computational challenges in bioinformatics. By analyzing the divergence observed in the molecular sequences of living species, phylogenetic procedures make possible the understanding of the evolutionary processes that defined the history of life. This kind of biological analyses can be modeled as an optimization problem which aims to reconstruct the evolutionary hypothesis that maximizes or minimizes a specific optimality criterion. Two examples of well-known optimality criteria in phylogenetics are the parsimony criterion [61] (Occam's razor) and the likelihood statistical measurement [57]. Finding the optimal phylogenetic history under this kind of optimality criteria shows an NP-hard complexity, as phylogenetic search engines must deal with a huge number of possible solutions which grows exponentially according to the number of species to be studied. Further difficulties are given by the fact that different criteria model different principles about the way species evolve in nature. That is, different methods make their own assumptions about the evolutionary process and, therefore, they often give rise to conflicting explanations about the evolution of the species under study, as reported by multiple researches in the literature [22, 32, 111, 152]. Multiobjective optimization represents an opportunity to deal with the conflicts that arise during the inference process. This approach provides mechanisms to obtain a compromise answer to the problem by inferring a set of high-quality solutions which are supported by different criteria simultaneously.

The formulation of phylogenetic inference as a MOP implies additional difficulties, as searches for multiple phylogenetic hypotheses must be performed over an exponentially-growing solution space in accordance with several time-consuming optimality criteria. The need to provide efficient multiobjective approaches to deal with this relevant biological problem motivates the research carried out in this PhD Thesis. Therefore, this Thesis aims to tackle the phylogenetic inference problem from a multiobjective perspective by means of bioinspired and parallel computing. The key idea is to conduct a detailed comparative study between different metaheuristic designs, assessing their

performance on real scenarios and exploring the implications of the multiobjective formulation from a biological point of view. These designs will be supported by parallel computing techniques in order to carry out efficient multiobjective searches in reasonable times. For this purpose, we will study parallel approaches based on different standard libraries, such as Open Multi-Processing (OpenMP) [29], Message Passing Interface (MPI) [74], and Open Computing Language (OpenCL) [68].

The interest of this Thesis is based on the following ideas. Firstly, advances in algorithmic approaches for phylogenetics can lead to significant contributions in those scientific domains where phylogenetic inference has practical applications. We can highlight areas such as evolutionary biology, biochemistry, physiology, ecology, paleontology, and medical research. In addition, a wide variety of real applications (including the study of molecular epidemiology patterns, drug discovery, and migration routines) can benefit from the knowledge provided by this research. Secondly, this Thesis aims to propose new algorithmic developments whose designs can be exported to address other optimization problems in real-world domains. Finally, the study of different parallel paradigms can highlight the benefits which imply the exploitation of current parallel architectures to deal with NP-hard optimization problems.

1.2 Main Goals

The main goal of this PhD Thesis lies on studying the benefits of applying multiobjective bioinspired computing and parallelism to address one of the most relevant problems in computational biology, the inference of phylogenetic hypotheses. In order to achieve this fundamental goal, this research undertakes the development of different algorithmic approaches which combine nature-inspired search strategies and parallel computing techniques to deal with the high execution times required to conduct phylogenetic analyses on real biological data sets. Focusing on the multiobjective treatment of the problem, different multiobjective metaheuristics will be examined with the ultimate goal of deciding what kind of algorithmic approaches lead to the most satisfying solutions attending to their multiobjective and biological performance, adapting well-known reference algorithms and proposing new bioinspired multiobjective designs. In addition, the implications of this multiobjective formulation of the problem will be explored by addressing real-world scenarios in which single-criterion methods provide conflicting explanations about the evolution of the input species. Due to the high complexity of the problem, parallel adaptations of the proposed metaheuristics will be implemented in order to highlight how the computing capabilities of modern hardware setups can be exploited to conduct this kind of computationally demanding biological analyses in reduced time.

The successful achievement of these goals will be closely related to the relative performance obtained by the analyzed algorithmic approaches with regard to the state-of-the-art. In order to assess the relevance of the algorithmic solutions proposed in this Thesis, comparisons with a wide range of phylogenetic tools from the literature will be performed. Another requirement is given by the fact of providing statistically reliable results. To address this issue, experimental results will

be evaluated under a statistical methodology which will allow us to provide reliable conclusions supported by statistical testing procedures.

In this Thesis, we have followed a set of well-defined steps to carry out the research. Such steps can be defined in the following terms:

1. **Study of the biological and computational basis of the problem**. This step involved the understanding of the natural mechanisms which motivate the evolution of species, along with the modelling of phylogenetic inference as an optimization problem and its main sources of computational complexity.

2. **Study of the state-of-the-art**. Due to the high relevance of this problem in the bioinformatics area, multiple researches have been conducted to explore it. This step of the research considered a literature review aimed at understanding the different strategies proposed by other authors to conduct phylogenetic analyses. Particularly, we focused on those proposals which addressed the problem by applying computational intelligence methods and parallelism.

3. **Study of the basis of bioinspired computing and parallelism**. In this step, we studied the main ideas behind the development of bioinspired algorithmic approaches, as well as the design decisions to be considered when tackling MOPs. In addition, the learning of standard libraries for parallel programming on different hardware architectures was carried out.

4. **Analysis, design, and implementation of multiobjective bioinspired metaheuristics**, modelling the problem in accordance with the knowledge acquired in the first three steps. Following the different development trends in multiobjective optimization, two main lines of designs were implemented: Pareto dominance-based schemes and quality indicator-based approaches.

5. **Experimental evaluation of the applied bioinspired metaheuristics**. Under a statistical framework, comparisons among the studied multiobjective algorithms were carried out. Multiobjective metrics were used to measure the performance of the examined designs when exploring phylogenetic solution spaces, along with biostatistical tests to verify the biological quality of the attained solutions with regard to the state-of-the-art.

6. **Analysis, design, and implementation of parallel approaches to phylogenetics**. On the one hand, parallel implementations of the proposed bioinspired metaheuristics for shared-memory systems and hybrid shared-distributed memory clusters were developed. On the other hand, heterogeneous computing on CPU+GPU systems was explored to take advantage of data parallelism opportunities to address time-consuming evaluation procedures.

7. **Experimental evaluation of the applied parallel designs**, using for this purpose different parallel metrics. Detailed analyses of run-times and parallel scalability studies were conducted to examine the main factors that impact the parallel performance of the implemented methods, making comparisons with other parallel schemes implemented in other phylogenetic tools.

8. **Conclusions and diffusion of the obtained results**, discussing the success of the applied designs on the basis of the observed biological, multiobjective, and parallel performance.

1.3 Main Contributions

This PhD Thesis aims to provide significant contributions to the bioinformatics, metaheuristics, and parallelism domains by defining new algorithmic developments to tackle the grand computational challenge that phylogenetic inference represents. More specifically, the main contributions of this research can be listed as follows:

1. A discussion on the biological implications associated to the multiobjective formulation of phylogenetic inference, showing how multiobjective optimization can be useful to deal with the conflicts reported by single-criterion methods in real-world scenarios. As case study, a widely-studied dataset of salamander mitochondrial DNA will be used to assess the biological relevance of the multiobjective approach.

2. The adaptation and development of six bioinspired multiobjective metaheuristics to solve the phylogenetic inference problem, defined in accordance with two of the main lines of algorithmic research in multiobjective optimization.

   (a) Pareto dominance-based methods:

      i. Multiobjective Artificial Bee Colony Algorithm (MOABC), a swarm intelligence algorithm inspired by the collective behaviour shown by honey bees in nature [95].
      ii. Multiobjective Firefly Algorithm (MO-FA), a swarm intelligence metaheuristic based on the bioluminescence of fireflies [190].
      iii. Fast Non-Dominated Sorting Genetic Algorithm II (NSGA-II), a reference multiobjective evolutionary algorithm proposed by Deb et al. [52].
      iv. Strength Pareto Evolutionary Algorithm 2 (SPEA2), a reference multiobjective evolutionary algorithm proposed by Zitzler et al. [196].

   (b) Quality indicator-based methods:

      i. Indicator-Based Multiobjective Bat Algorithm (IMOBA), a swarm intelligence algorithm built upon the echolocation capabilities shown by bats [191].
      ii. Indicator-Based Evolutionary Algorithm (IBEA), the implementation of a general framework proposed by Zitzler and Künzli to integrate the computation of multiobjective quality indicators into algorithmic search engines [195].

3. Statistically-reliable comparative studies of multiobjective performance among the proposed metaheuristics on real nucleotide data sets, examining the benefits of developing advanced bioinspired search engines to address this kind of difficult optimization problems.

4. An analysis on the parallelization of the applied metaheuristics on shared-memory environments with OpenMP. Starting from synchronous parallel approaches, we give account of the advantages and disadvantages of each algorithmic design, also proposing a novel parallelization scheme for MOABC inspired by the asynchronous parallel behaviour of honey bees.

5. The application of MPI+OpenMP mixed mode programming to parallelize three representative multiobjective metaheuristics (MO-FA, NSGA-II, and IBEA) on hybrid shared-distributed memory multicore clusters, identifying and addressing the main issues that impact the performance of hybrid parallel systems.

6. The development of heterogeneous CPU and GPU strategies with OpenCL to accelerate time-consuming evaluation procedures in phylogenetics, considering the parallelization of the phylogenetic parsimony function as case study.

7. A complete assessment of parallel and biological performance with regard to a wide-variety of multiobjective and biological methods from the state-of-the-art, using for this purpose biological and statistical tests to ensure the reliability of the obtained conclusions.

1.4 Thesis Organization

This PhD Thesis is organized in a set of well-differentiated blocks of chapters. The first block (chapters 1-4) introduces the motivation behind this research, explaining the basis of the problem and giving account of related works proposed in the literature. It also details the methodology followed throughout the development of this Thesis. The second block (chapters 5-6) is focused on detailing the bioinspired metaheuristics applied to solve the problem, reporting the multiobjective and biological results obtained by experimentation on real biological data sets. The third block (chapters 7-10) assesses the parallel strategies adopted to address the computational complexity of the problem by exploiting the capabilities of current hardware architectures. The final block (chapters 11-12) contains the main conclusions of this Thesis, defines future work directions, and reports scientific achievements.

The different chapters which compose this document are briefly explained below:

1. **Chapter 1 - Introduction**. This chapter introduces the main challenges which motivate the study of phylogenetic relationships by means of parallel and bioinspired computing, defining the scope of the research and the key goals to be attained. Moreover, it summarizes the main contributions of this Thesis and presents the organization of this document.

2. **Chapter 2 - Phylogenetic Inference**. This chapter firstly studies the biological mechanisms that define the history of evolution. Afterwards, it introduces a computational modelling of the problem, describing the input data required to conduct phylogenetic analyses and how the evolution is represented by means of tree-shaped phylogenies. In this chapter, we also address

the formulation of phylogenetic inference as a MOP, detailing the objective functions to be employed and the main computational issues to be considered.

3. **Chapter 3 - Related Works**. This chapter contains a literature review on the topic in this Thesis. Researches from other authors are described, focusing specially on those developments which applied approximated methods and metaheuristics to address the problem. Special treatment is given to the study of high performance computing techniques for phylogenetics, highlighting the most relevant parallel proposals reported in the literature.

4. **Chapter 4 - Methodology**. In this chapter, we report the methodology followed throughout this Thesis. First of all, we introduce the basis of multiobjective bioinspired computing, defining some notions about the modelling of MOPs and algorithmic development. An explanation of the different parallel computing paradigms considered in this research is introduced afterwards. This chapter also details the different methods used to assess the proposed metaheuristics: multiobjective quality indicators, biological testing procedures, and parallel performance metrics. This chapter ends with the introduction of our experimental and statistical methodology and the description of materials and data sets.

5. **Chapter 5 - Multiobjective Metaheuristics for Phylogenetic Inference**. This chapter is focused on describing the different multiobjective metaheuristics applied to solve the problem. It distinguishes between algorithmic approaches based on Pareto dominance (MOABC, MOFA, NSGA-II, and SPEA2) and approaches based on the integration of quality indicators into multiobjective search (IMOBA and IBEA). For each metaheuristic, detailed pseudocodes and explanations about their designs are provided.

6. **Chapter 6 - Multiobjective and Biological Results**. This chapter undertakes the evaluation of the solutions reported by each metaheuristic from both multiobjective and biological perspectives, assessing result quality by reporting comparisons with other authors' methods. This chapter also provides insight into the biological implications of conducting multiobjective phylogenetic analyses, using a well-known case study to show the benefits of the proposed formulation of the problem.

7. **Chapter 7 - Metaheuristic Parallelization Approaches**. In this chapter, the parallelization of multiobjective metaheuristics for phylogenetics is explained. Firstly, OpenMP-based designs for shared-memory environments are detailed, distinguishing between synchronous generational parallelization schemes and asynchronous non-generational models. Secondly, cluster computing techniques for shared-distributed memory hybrid systems are explored, providing MPI+OpenMP parallelization schemes based on mixed mode programming.

8. **Chapter 8 - Performance Assessment of Parallel Metaheuristics**. The parallel performance of our metaheuristic designs is examined in this chapter. For this purpose, parallel scalability studies on different problem and system sizes are described, along with detailed run-time

analyses and the evaluation of speedups and efficiencies. Comparisons with other parallel approaches to phylogenetics are introduced to verify the relevance of our parallelization schemes.

9. **Chapter 9 - Heterogeneous Approaches for Phylogenetics**. In this chapter, we explore how phylogenetic methods can benefit from the computing capabilities of heterogeneous GPU+CPU systems, considering the parallelization of the phylogenetic parsimony function with OpenCL. GPU and CPU kernel implementations of this relevant function are described, along with a multidevice design which simultaneously orchestrates CPU and GPU devices to take full advantage of the underlying computing devices.

10. **Chapter 10 - Performance Assessment of Heterogeneous Approaches**. This chapter evaluates the parallel performance of our heterogeneous implementations of the parsimony kernel. The speedups observed on single and multidevice system configurations are reported and discussed, pointing out how the characteristics of the input data govern the performance obtained by each computing device. Comparisons with other parallel proposals are included to evaluate speedup values with regard to state-of-the-art implementations of the parsimony function.

11. **Chapter 11 - Conclusions**. In this chapter, we include concluding remarks in accordance with the results presented and discussed in the previous chapters. On the basis of these conclusions, we propose the opening of new lines of research that can be undertaken in future works.

12. **Chapter 12 - Scientific Achievements of the Research**. This final chapter highlights the scientific publications achieved as a consequence of the relevant results obtained in this Thesis. Other significant achievements obtained throughout the development of this Thesis are also pointed out, including the participation of the PhD candidate in research projects, international stays, organization of conferences, workshops, and special sessions, reviewer tasks for international journals, teaching tasks, and collaborations with other institutes.

1.5 Summary

This chapter has defined the main scope of this PhD Thesis, which aims to apply new algorithmic designs based on bioinspired and parallel computing to tackle one of the most relevant problems in bioinformatics: phylogenetic inference. Identifying the main issues to be addressed, we have given account of the motivation of the research and the main goals to be attained. The main contributions of this research include the application of up to six multiobjective metaheuristics defined to carry out phylogenetic analyses, proposing three new bioinspired designs and adapting three well-known reference algorithms for solving MOPs. In addition, high performance computing techniques are proposed to parallelize these algorithms on shared-memory multiprocessor systems and hybrid shared-distributed memory cluster platforms, exploring further developments based on heterogeneous CPU+GPU computing to address time-consuming operations. Finally, the organization of this document has been introduced.

Chapter 2

Phylogenetic Inference

This chapter introduces the problem tackled in this PhD Thesis: the inference of phylogenetic histories. Firstly, we briefly define some key concepts on the development of bioinformatics applications, pointing out how biological processes are currently modelled and addressed as optimization problems. Afterwards, we focus on detailing the basis of phylogenetic inference, explaining the evolutionary mechanisms which are studied in this kind of biological analyses. After defining inputs and outputs, phylogenetic methods are classified and the objective functions to be considered in this research are formulated. Finally, we discuss the biological implications of multiobjective optimization in phylogenetics and the computational complexity of the problem.

2.1 Optimization Problems in Bioinformatics

Bioinformatics is a multidisciplinary field which aims to combine computer science techniques, algorithmic development, and statistical procedures to solve major problems in molecular biology. More specifically, the bioinformatics area is focused on the application of computational methods to model biological systems at the atomic, molecular, metabolic, cellular, and pathologic levels [167]. The problems tackled in bioinformatics involve the analysis, storage, manipulation, and interpretation of molecules which codify biological information, such as DNA, RNA, and amino acids. By studying these structures of life, the information contained in these molecules can be interpreted, leading to advances in the understanding of biological processes in nature.

Representative examples of biological problems which are subject of study by bioinformaticians include DNA mapping and sequencing, gene finding and pattern matching in DNA sequences, protein structure and function prediction, sequence alignment, and the reconstruction of evolutionary histories from molecular data of living species. In order to process the huge amount of biological information codified at the molecular level, bioinformatics applications are built upon the basis of different computer science disciplines, such as machine learning, information theory, graph theory, artificial intelligence, stochastic methods, high performance computing, and so on.

The development of bioinformatics applications involves two well-defined steps:

1. The modelling of the biological problem to be addressed, formulating biological processes in a computationally tractable way.

2. The design of an algorithm which solves the computational problem associated to the biological reality formulated by the constructed model.

Focusing on the first step, an accurate modelling of the problem requires deep knowledge on the underlying biological mechanisms. Such mechanisms are often governed by some kind of biological principle which assesses the quality of the generated biological structures. In this way, different structures can be compared attending to their biological quality, giving preference to the fittest ones. Therefore, biological processes often resemble optimization processes which are suitable to be mathematically formulated and implemented in a computer application. This is the reason why a wide variety of problems encountered in bioinformatics are usually tackled as optimization problems [132]. Formally, an optimization problem is defined in the following terms:

Definition 2.1.1. Optimization problem. An optimization problem consists of exploring a search or decision space SS with the aim of finding the solution $s \in SS$ which optimizes (maximizes or minimizes) an objective function $f : SS \rightarrow \Re$ [176].

$$\begin{aligned} &\text{Optimize } f(s) \in \Re \\ &\quad \text{where } s = [s_1, s_2, ..., s_k] \in SS. \\ &\text{Such that } g_i(s) \leq 0, i = 1, ..., p, \\ &\qquad h_j(s) = 0, j = 1, ..., q. \end{aligned} \tag{2.1}$$

In the above expression, $s \in SS$ is codified by means of k decision variables $[s_1, s_2, ..., s_k]$ that represent the numerical quantities which must be explored throughout the optimization process. Most optimization problems impose certain restrictions according to the nature of the modelled problem. These restrictions, expressed in the shape of mathematical inequalities ($g_i(s)$) and equalities ($h_j(s)$), are commonly known as constraints. Those solutions that satisfy these constraints belong to the space of feasible solutions to the problem. The objective function $f(s)$ defines the optimization goal to be achieved. This function associates a solution in SS with a real value that identifies solution quality in accordance with the objective to be optimized.

Traditionally, biological optimization problems have been addressed by considering a single objective function to be maximized or minimized. However, biological processes are often conducted following not only a single principle, but a number of different optimality criteria which govern the behaviour of real natural systems. That is, biological procedures usually involve the optimization of multiple objective functions simultaneously. This reasoning justifies the current interest in modelling bioinformatics problems as multiobjective optimization problems [77]. This kind of problems can be formally defined in the following way:

Definition 2.1.2. Multiobjective optimization problem. A multiobjective optimization problem (MOP) involves finding a set of solutions s characterized in a decision space SS which optimize simultaneously n objective functions $\vec{f}(s)$ defined in an objective space $Z = \Re^n$, where $n \geq 2$ [36].

$$\begin{aligned} &\text{Optimize } z = \vec{f}(s) = [f_1(s), f_2(s), ..., f_n(s)], \\ &\quad \text{where } s = [s_1, s_2, ..., s_k] \in SS, \\ &\quad\quad z = [z_1, z_2, ..., z_n] \in Z. \\ &\text{Such that } g_i(s) \leq 0, i = 1, ..., p, \\ &\quad\quad h_j(s) = 0, j = 1, ..., q. \end{aligned} \tag{2.2}$$

A MOP considers two multidimensional spaces: a k-dimensional decision space SS, which defines the bounds of each decision variable, and an n-dimensional objective space $Z = \Re^n$, representing the ranges of the objective functions to be optimized. Every point in SS that satisfies the constraints g_i and h_j represents a feasible solution to the MOP. In addition, for each solution in SS, there is a corresponding point in Z represented by the objective vector z, which measures the quality of the solution according to each one of the considered objectives.

As the optimization process in a MOP considers multiple and often conflicting objective functions, multiobjective optimizers seek to find not a single solution (as in traditional single-objective optimization), but a set of solutions named as Pareto-optimal solutions. This set represents the best possible trade-offs among the considered objectives. Therefore, the main goal is to find high-quality compromises among the objective functions to be optimized. From a bioinformatics perspective, the application of multiobjective optimization techniques is becoming increasingly popular, obtaining significant benefits over traditional single-objective proposals for biological system optimization, classification, and inverse problems [77]. Furthermore, a wide spread of bioinformatics problems have been successfully addressed by means of multiobjective optimization techniques. Examples of this kind of researches include developments for cancer therapy discovery [112], gene regulatory networks [101, 168], sequence and structure alignment [130, 179], structure prediction and design [43, 47, 50], microarray data analysis and profiling [10, 82, 97], and DNA computing [160].

Regarding the developments of algorithmic designs for bioinformatics, a key feature that characterizes these biological problems is given by the vast amount of data to be processed. Throughout the years, the biological information available for research has grown exponentially thanks to the advances in molecular sequencing. Thereby, solving biological optimization problems by means of traditional exhaustive searches becomes infeasible, requiring exponential time even in low-sized data sets. The NP-hard nature of a wide range of bioinformatics applications represents a challenging problem from a computational perspective [167]. The literature has tackled such complexity by proposing algorithmic designs built upon the basis of computational intelligence, taking a very significant role the development of bioinspired metaheuristics for bioinformatics [132]. In this context,

the exploitation of current parallel architectures has also taken a major role in the bioinformatics area. In order to address increasing problem sizes, bioinformaticians have focused their research efforts on taking advantage of parallelism opportunities at the molecular processing level, benefiting from reduced execution time or improved search capabilities in accordance with the way parallel computing techniques are integrated into the algorithmic design [198].

2.2 Basis of Phylogenetic Inference

Phylogenetic inference is one of the most representative problems in the field of bioinformatics. The study of the evolution of species has a significant importance not only from a biological point of view, but also from a computer science perspective, as it represents a grand computational challenge [9] whose solution represents a difficult task [40]. This section aims to explain the basis of phylogenetic inference, detailing its formulation as a MOP and discussing its computational complexity.

2.2.1 Understanding the Biological Mechanisms Behind Evolution

The phenotypic features observed in living organisms are a reflection of the genetic information contained in their molecules and their interactions with the environment. The information that organisms carry on is codified in their DNA molecules, which are composed of different nucleotide bases, named as purines (adenine and guanine) and pyrimidines (cytosine and thymine, the former being replaced by uracil in RNA).

DNA/RNA polymerases are enzymes which take a key role in the creation of DNA molecules, as they are in charge of assembling nucleotides. These enzymes can introduce non-complementary nucleotides during gene duplication that modify the genetic information codified in these molecules. The result of this process is generally called a *mutation*. Attending to the types of nucleotide bases involved, mutations can be classified into *transitions* (from a purine to a purine or from pyrimidine to pyrimidine) and *transversions* (from purine to pyrimidine and vice versa). The rate at which mutations arise at the DNA/RNA level is called *mutation rate*. Other mechanisms that modify genetic information include genetic exchanges (recombination), deletions or insertions of one of more nucleotides (indels), gene duplication, etc. When this genetic information is altered, the codified proteins may have altered functions, giving rise to different phenotypic characteristics observed in the organism. If the organisms resulting from these mutations are fitter than other ones in the environment, they will be subject to positive selective pressure as an *advantageous mutation* has taken place. Otherwise, negative selective pressure will be applied over them, as the mutation has turned into a *disadvantage* for the affected organism. *Neutral mutations* can also take place and evolve into advantageous or disadvantageous ones according to possible changes in the environment.

Those mutations that are passed on from parents to their offspring and coexist with the original gene are called *polymorphisms*. In this context, multiple variants of a gene can be found simultaneously within a population. The specific gen in the genome which shows these multiple variants

is known as *locus*, while the variants themselves are known as *alleles*. Throughout the generations, some alleles will increase their frequency in the population until all their individuals show the same allele in the locus. When this situation happens, we say that such alleles have become fixed in the population. As a consequence, the resulting population might show some significant differences with regard to the ancestral population that allow their organisms to be classified as new species. The rate at which populations diverge depends on the mutation rate, the time between two generations or generational time, and different evolutionary forces (such as fitness of the alleles, population size, selective pressures). In this sense, the term *substitution* refers to the differences observed among multiple organisms at the molecular sequence level.

Given two populations of a single species, an evolution into two different children species will take place if they lose the capacity to interbreed, as a result of the accumulation of substitutions that avoid a successful mating. Under this assumption, speciation mechanisms are classified according the evolutionary forces that govern them. The three more important classes of speciation are allopatric speciation, sympatric speciation, and parapatric speciation [105]:

- **Allopatric speciation** refers to the evolution of a parent population into different children species due to geographical isolation. As a result of habitat fragmentation (rise of a new geographic barrier, migration beyond a geographical barrier, etc), gene flow between populations cannot take place. The isolated populations will be affected by different evolutionary mechanisms (mutations, selective pressures) over time, giving rise to significant genotypic and phenotypic divergence, along with losing the capacity to exchange genes. This kind of speciation represents one of the most common processes which lead to the birth of new species.

- **Sympatric speciation** defines the evolution of a population into different children species within the same geographic location. In order to observe this kind of speciation, the magnitude of the positive selective pressure associated to mutations should be very large, so that a fast morphological change preventing the capacity to interbreed takes place. It is considered an uncommon but plausible speciation mechanism which explains the emergence of new species from a single parent population in the same geographical region.

- **Parapatric speciation** lies between allopatry and sympatry. In this class, the rise of new species takes place when populations showing multiple alleles are partially separated from each other. Although the geographic location allows partial contact between populations, different selection pressures support an unequal gene flow which increases differences from one population to another. Those mutations which specifically prevent successful inter-breeding represent a key factor behind parapatric speciation.

2.2.2 Phylogenetic Methods

The biodiversity currently observed in nature reflects the effects of a wide variety of evolutionary mechanisms which allowed species to obtain different morphological characteristics throughout the

```
5  15
organism1        T T C A A T C A G G C C C G A
organism2        T C A C C A A T A G A C T G A
organism3        T C A A G T C A G G T T C G A
organism4        T A C C A A T A G A C T C C A
organism5        T C C A G T T A G A C T C G A
```

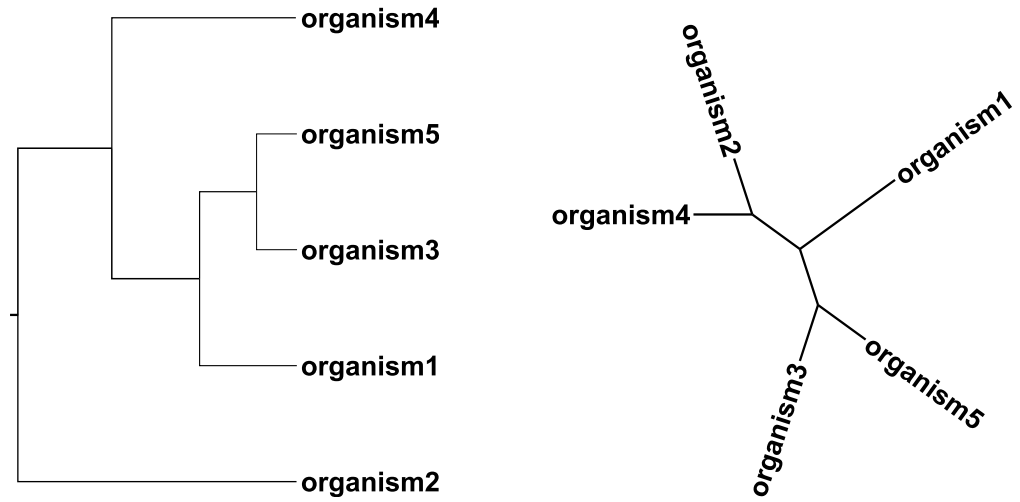
Fig. 2.1 Example of input sequence alignment codified in Phylip format

years. Molecular phylogenetics is the field of computational biology focused on studying genealogical relationships among related organisms from molecular data, with the ultimate goal of reconstructing the evolutionary history of all organisms in earth, the *tree of life* [46]. Phylogenetic methods seek to combine statistics, computational techniques, and algorithmic procedures in order to give support to the study of evolution, identifying divergence at the molecular level to infer ancestral-descendant relationships among organisms [105].

Inputs and Outputs

The input of phylogenetic methods is given by an $N \times M$ matrix of characters representing molecular sequences which describe biological information about living organisms, where N is the number of species under study and M the number of characters or sites per sequence. The sequences processed in a phylogenetic analysis can be associated to different molecular features such as DNA, RNA, amino acid, or other biological data. In accordance with the type of molecular data studied, each character in the sequences takes different state values defined by an alphabet Λ. Particularly, DNA-based inferences take as input nucleotide sequences from those species whose ancestral relationships must be inferred, represented by means of the alphabet of nucleotide bases $\Lambda=\{A,C,G,T\}$, which is usually complemented by considering the presence of gaps (-), missing or unknown states (?), and other information (for example, combinations of nucleotides). In order to conduct phylogenetic analyses, the input sequences must be aligned, so the original data must be previously processed by applying sequence alignment methods. Figure 2.1 shows an example of an alignment composed of 5 nucleotide sequences of length 15, codified under the widely-used Phylip format [59]. This format defines a header composed of two numerical quantities: the number of species and the length of their aligned sequences. The body of the file defines the identifiers of the species under study followed by their related nucleotide sequences, using a separate line for each organism.

In phylogenetics terminology, those modern species whose genetic information is available are referred as *operational taxonomic units* (OTUs). This information is analyzed to verify substitution patterns among organisms, identifying those mutation events that led to the diversity observed at the



(a) Rooted tree (b) Unrooted tree

Fig. 2.2 Examples of phylogenetic topologies

nucleotide level. In this way, we can define their evolutionary relationships through the definition of common hypothetical ancestors. Such ancestral organisms are identified by the term *hypothetical taxonomic units* (HTUs). Under these assumptions, the course of evolutionary history is expressed by using tree-shaped structures $\tau = (V, E)$ known as *phylogenetic trees* (or simply *phylogenies*). In these structures, the node set V contains the different species whose evolutionary history is described, locating OTUs at the leaves of the tree while representing HTUs as internal nodes. The branch set E defines genealogical linkages between related nodes, whose branch length values quantify generational times from ancestors to descendants.

In accordance with the observed branching patterns, different groups of nodes can be distinguished in a phylogenetic tree. On the one hand, a group of nodes which includes all the organisms descended from a single ancestor is called a *monophyletic* group. On the other hand, a *polyphyletic* group is composed of unrelated organisms descended from more than one ancestor. Finally, a *paraphyletic* group is the one which contains the most recent common ancestor, but some of the descendants are missing. In addition, the definition of a global common ancestor which represents the origin of the phylogeny allows classifying phylogenetic trees into rooted and unrooted topologies. A rooted topology contains a root node which indicates the direction of evolution in the phylogeny, representing the ancestor of all OTUs. When this root node does not exist, the direction of the evolutionary process cannot be defined and we say that the tree shows an unrooted topology. Figure 2.2 shows examples of rooted and unrooted phylogenetic topologies.

Classification of Methods

The study of evolutionary histories can be carried out by using different approaches. One of the most well-known classification criteria distinguishes phylogenetic methods according to the kind of

data used as basis to infer phylogenetic relationships. Under this criterion, phylogenetic procedures are grouped into character-based methods and distance-based methods [105]. On the one hand, a *character-based method* operates directly over the discrete character states described in the input alignment. This class of methods usually proceeds by generating a starter tree, for example, by grouping the organisms in the order they appear in the sequences. Afterwards, they perform a topological search based on the study of divergence and similarities observed at the character state level, modifying the starter tree until a satisfying phylogenetic topology has been retrieved. In order to assess the quality of the phylogeny, character-based optimality criteria are often used to guide the topological search towards high-quality phylogenetic topologies attending to some biological and statistical principles. Examples of character-based approaches include the maximum parsimony [61] and maximum likelihood [57] methods.

On the other hand, *distance-based methods* are focused on obtaining phylogenetic histories from the processing of an *NxN* symmetric matrix containing evolutionary distances, where N is the number of species in the input alignment. Each entry i, j in a distance matrix defines a floating point measurement of the pairwise genetic or evolutionary distance observed between two OTUs i and j in the sequence alignment. A first approach to measure genetic distances consists of counting the fraction of positions in which differences are found between each pair of sequences. This quantity, known as p-distance, is usually complemented by considering probabilistic measurements given by mathematical models of molecular evolution [16], resulting into the evolutionary distances which will be processed by a distance-based approach. Once evolutionary distances have been defined, the distance-based method builds the phylogeny by grouping species according to the measured distances. Examples of distance-based approaches are the unweighted pair group method with arithmetic means (UPGMA) [165], the weighted pair group method with arithmetic means (WPGMA) [163], and the neighbour-joining (NJ) [149] methods. Distance-based optimality criteria may also be used to perform topological searches, as implemented in the Fitch-Margoliash approach [62].

The mathematical models used to incorporate biostatistical information into the inference process are known as *evolutionary models*. Those phylogenetic methods which conduct the inference process according to the assumptions given by an evolutionary model are denoted as *model-based methods*. More in detail, evolutionary models characterize the probabilities of observing substitution events from one character state to another. For this purpose, an evolutionary model defines an instantaneous rate Q *matrix* which specifies the relative rates of change for each nucleotide. Figure 2.3 shows an example of Q matrix for one of the most widely-used models of nucleotide evolution, the General Time Reversible model (GTR) [145]. In this model, each entry in the Q matrix represents the instantaneous substitution rate from a nucleotide i to a nucleotide j, being μ the mean instantaneous substitution rate, a, b, c, d, e, and f the relative rates of each nucleotide substitution to any other, and π_A, π_C, π_G, π_T the stationary probability for each nucleotide base.

On the basis of the instantaneous rate Q matrix, it is possible to define the probabilities of observing a mutation event within an evolutionary time t as $P(t) = exp(Qt)$. The simplest nucleotide evolutionary model is given by the Jukes-Cantor proposal (JC69) [94], which considers a Q matrix

	A	C	G	T
A	$-\mu(a\pi_C+b\pi_G+c\pi_T)$	$a\mu\pi_C$	$b\mu\pi_G$	$c\mu\pi_T$
C	$a\mu\pi_A$	$-\mu(a\pi_A+d\pi_G+e\pi_T)$	$d\mu\pi_G$	$e\mu\pi_T$
G	$b\mu\pi_A$	$d\mu\pi_C$	$-\mu(b\pi_A+d\pi_C+f\pi_T)$	$f\mu\pi_T$
T	$c\mu\pi_A$	$e\mu\pi_C$	$f\mu\pi_G$	$-\mu(c\pi_A+e\pi_C+f\pi_G)$

Fig. 2.3 Q matrix for the General Time Reversible evolutionary model

with π_A=π_C=π_G=π_T=1/4 and a=b=c=d=e=f=1. Under these assumptions, the probability of remaining in the same character state ($P_{ii}(t)$) and the probability of a change from a nucleotide i to another one j ($P_{ij}(t)$) are given by the following expressions [105]:

$$P_{ii}(t) = {}^1\!/_4 + {}^3\!/_4 \exp(-\mu t) \tag{2.3}$$

$$P_{ij}(t) = {}^1\!/_4 - {}^1\!/_4 \exp(-\mu t) \tag{2.4}$$

The correction of evolutionary distances can be obtained from these equations, defining μt as $-1/2log(1-4/3p)$, where p is the p-distance observed between two organisms in the input alignment. By substituting μt with $2/3d$ (being d the evolutionary distance to be computed), the Jukes-Cantor distance correction formula is obtained:

$$d = -{}^3\!/_4 \ln(1 - {}^4\!/_3 p) \tag{2.5}$$

Figure 2.4 shows an example of the two main methodologies in phylogenetic reconstruction. On the one hand, the character-based method generates from the input alignment a simple starter phylogenetic topology by grouping the organisms in accordance with the order they appear in the input dataset (*org1* with *org2*, *org3* with *org1+org2*, and so on), defining a common ancestor at each step. From the starter tree, an optimality criterion is used to guide a topological search which performs changes in the branching patterns, leading to a satisfying phylogenetic tree according to the used criterion. This topological search can be conducted under evolutionary model assumptions (as in the case of maximum likelihood) or not (as in maximum parsimony searches). More details on the topological search step will be introduced later in Section 2.2.4.

On the other hand, the distance-based methodology generates a p-distance matrix from the input alignment by computing the fraction of changes observed in the molecular sequences of each pair of organisms. In this example, the Jukes-Cantor correction (Equation 2.5) is applied to generate evolutionary distances. The resulting distance matrix is then taken as input by a distance-based

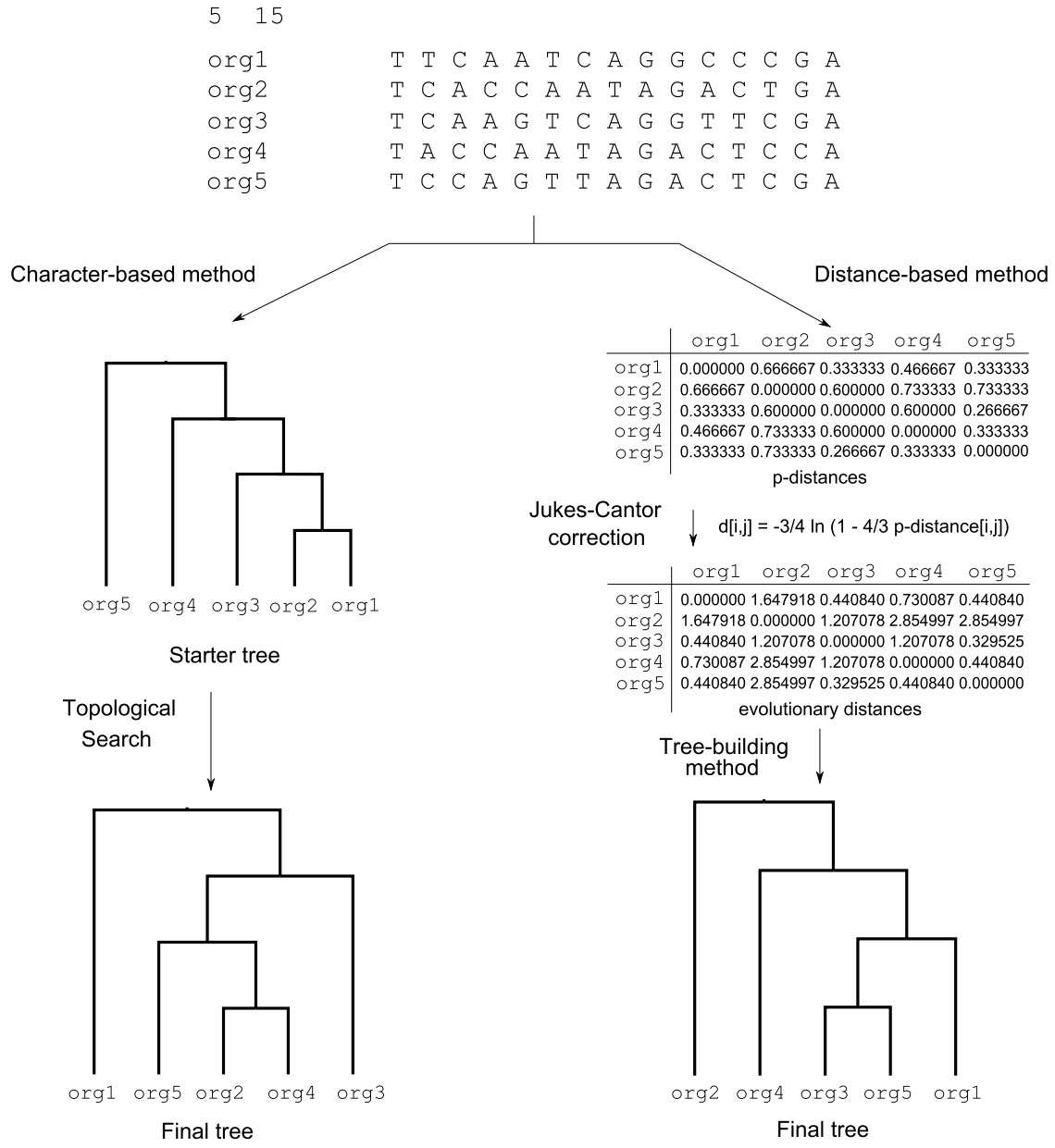


Fig. 2.4 Example of phylogenetic inference under character-based and distance-based methods

tree-building method, for example, a clustering method [105], that generates the corresponding phylogenetic topology. More details on distance-based methods will be introduced later in Section 2.2.4.

Certain state-of-the-art implementations of phylogenetic methods merge these two methodologies to improve their search capabilities. For example, the combination of methodologies may take place at the starter tree generation in a character-state method. Instead of adding species in the order they appear or randomly, a distance method is applied to generate a good starting point for a phylogenetic search engine following a character-based optimality criterion. Therefore, the combination of both classes of methods can be useful to enhance the potential of the phylogenetic procedure.

2.2.3 Optimality Criteria

As introduced previously, modern phylogenetic methods rely on the definition of an optimality criterion which specifies a measurement of biological quality to guide the inference process. The main goal is to find the fittest solution according to the implemented optimality criterion. Under this perspective, phylogenetic inference is addressed as an optimization problem, consisting of exploring a phylogenetic tree search space SS with the aim of finding the phylogeny $\tau = (V,E) \in SS$ that optimizes an objective function $f(\tau)$, that is, the evolutionary hypothesis best supported by the applied optimality criterion. The scope of this Thesis considers two of the most well-known character-based objective functions in the phylogenetics area: maximum parsimony and maximum likelihood.

Parsimony criterion

Among different competing hypotheses that explain the nature of a system, Occam's razor suggests that the simplest one should always be preferred. This statement is a well-known problem-solving principle that is followed in a wide range of scientific domains, including bioinformatics. Maximum parsimony approaches apply Occam's razor as the basic criterion behind the inference of evolutionary relationships, giving preference to the simplest evolutionary history. The maximum parsimony problem can be defined as follows:

Definition 2.2.1. Maximum parsimony problem. Given a set of N aligned molecular sequences composed of M sites per sequence from different organisms, the maximum parsimony problem aims to find the phylogenetic tree $\tau = (V,E)$ that minimizes the amount of substitutions observed between related organisms throughout its topology. Such topology represents the most parsimonious evolutionary hypothesis, that is, the simplest explanation to the evolutionary process which led to the molecular features described in the input data. The parsimony value $P(\tau)$ for a phylogenetic tree $\tau = (V,E)$ is computed in the following way [105]:

$$P(\tau) = \sum_{i=1}^{M} \sum_{(a,b)\in E} C(a_i, b_i), \tag{2.6}$$

where $(a,b) \in E$ represents the ancestral relationship which connects two nodes $a,b \in V$, and $C(a_i,b_i)$ measures the divergence observed between a and b at the i-th character. This cost function can be calculated in the following way:

$$C(a_i,b_i) = \begin{cases} 1 & \text{if } a_i \neq b_i, \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

Previously to the calculation of the phylogenetic parsimony function, we must define an assignment of character state sets S per node that reduces the number of changes observed throughout the topology. This task is often carried out by applying a two-step procedure known as Fitch's algorithm [61]. The first step of this algorithm specifies a bottom-up strategy which computes, for each

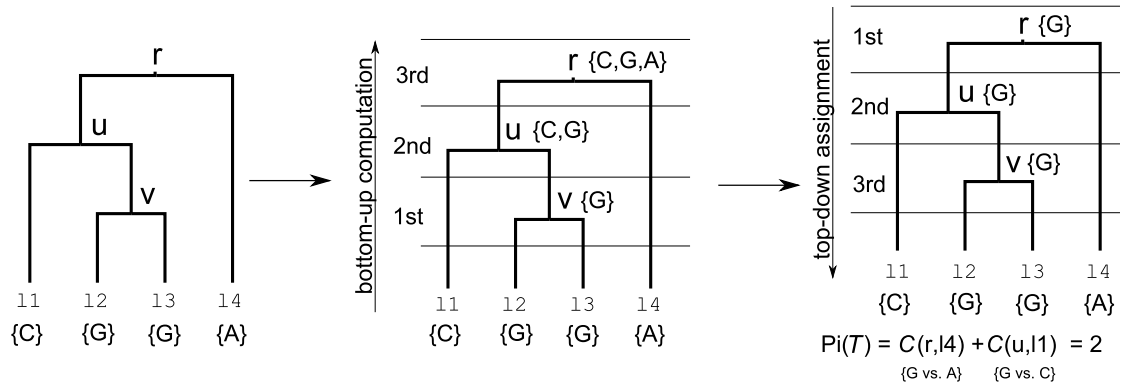


Fig. 2.5 Example of phylogenetic inference under maximum parsimony: application of Fitch's algorithm and parsimony computations

character i in the sequences, a set of possible states S_i to each node in the tree (proceeding from the leaves to the root). For an OTU $l \in V$, the state set $S_i(l)$ is given by the value l_i observed at the i-th character of the input sequence corresponding to the organism l. Once state sets for the leaves have been defined, ancestral state sets for the internal nodes in the topology are calculated. Given a HTU denoted by an internal node $u \in V$ with children $v, w \in V$, $S_i(u)$ is computed as follows:

$$S_i(u) = \begin{cases} S_i(v) \cap S_i(w) & \text{if } S_i(v) \cap S_i(w) \neq 0, \\ S_i(v) \cup S_i(w) & \text{if } S_i(v) \cap S_i(w) = 0. \end{cases} \tag{2.8}$$

After defining the sets S_i, the second step in Fitch's algorithm computes the assignment of final character states by following a top-down strategy. Starting from the root node r, the state r_i is defined by choosing any state value contained in $S_i(r)$. For each remaining HTU v with ancestor u, v_i takes the value u_i iff u_i is included in $S_i(v)$. Otherwise, any state value contained in $S_i(v)$ is chosen as v_i.

Figure 2.5 shows a simple one-character example of parsimony computation for a phylogenetic tree composed of four OTUs ($l1$, $l2$, $l3$, $l4$) and 3 HTUs (the root r, u, and v). The possible state sets for each HTU are defined during the bottom-up step as $S_i(v) = \{G\}$ (as its children $l2$ and $l3$ share the same character state 'G'), $S_i(u) = \{C, G\}$ (as $l1$ and v do not show any common state value), and $S_i(r) = \{C, G, A\}$ (as u and $l4$ do not share any state value). The top-down assignment step proceeds by assigning $r_i = \{G\}$, $u_i = \{G\}$ (as 'G' is contained in $S_i(u)$), and $v_i = \{G\}$ (also contained in $S_i(v)$). In accordance with Equations 2.6 and 2.7, the parsimony score of the topology in this character is 2, as different state values are observed in the branches (r, $l4$) and (u, $l1$).

Likelihood criterion

The likelihood function is a well known statistical measurement that defines the conditional probability of observing an outcome D given a certain hypothesis Θ. Applying this function to phylogenetics, D refers to the input aligned sequences, and Θ represents an evolutionary history modelled by a phylogenetic topology and a model of nucleotide evolution. More in detail, the phylogenetic likelihood

function measures the probability that the evolutionary history suggested by a model-based method gives rise to the diversity observed in the input alignment. Therefore, the maximum likelihood problem can be defined in the following terms:

Definition 2.2.2. Maximum likelihood problem. Given N aligned molecular sequences from different organisms, with M characters per sequence, the maximum likelihood problem consists of finding the most likely evolutionary hypothesis which explains the observed data. Such hypothesis is described by a phylogeny $\tau = (V,E)$ inferred under the assumptions given by an evolutionary model Φ. Among different competing evolutionary hypotheses, the one that maximizes the likelihood statistical measurement will be preferred. Under the probabilistic framework given by the evolutionary model Φ , the likelihood value $L(\tau,\Phi)$ for a phylogenetic tree $\tau = (V,E)$ is computed as [20]:

$$L(\tau,\Phi) = \prod_{i=1}^{M} \sum_{x\in\Lambda} \pi_x L_p(r_i = x), \tag{2.9}$$

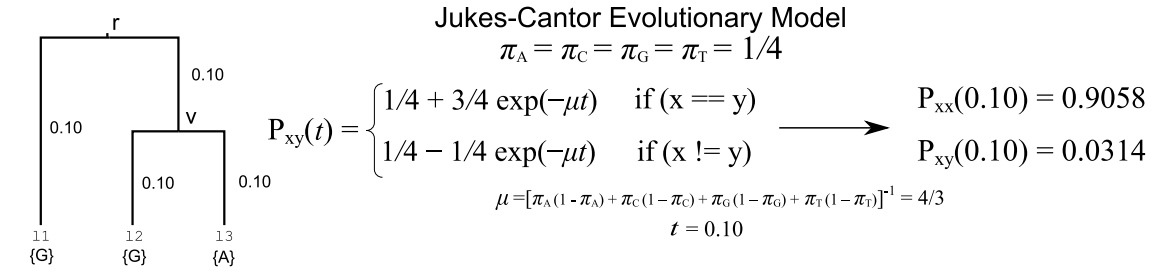
where Λ represents the character state alphabet, π_x is the stationary probability associated to the state $x \in \Lambda$, and $L_p(r_i = x)$ the partial likelihood of observing x at the i-th character of the sequences associated to the root node r of the tree. This partial likelihood score can be computed in a recursive approach in accordance with the position of each node in the topology, following Felsenstein's pruning algorithm [57]. The likelihood $L_p(u_i = x)$ for a HTU $u \in V$ with children nodes $v,w \in V$ is defined by the following expression:

$$L_p(u_i = x) = \left(\sum_{y\in\Lambda} P_{xy}(t_{uv}) L_p(v_i = y) \right) \times \left(\sum_{y\in\Lambda} P_{xy}(t_{uw}) L_p(w_i = y) \right), \tag{2.10}$$

where t_{uv} and t_{uw} refer to the generational times from u to its children v and w, described by the length values of the branches (u,v), $(u,w) \in E$, and $P_{xy}(t)$ represents the probability of observing a mutation event from the state x to y within a time t according to the chosen evolutionary model. Regarding the OTUs case, the partial likelihood $L_p(l_i = x)$ for a leaf node $l \in V$ is computed as:

$$L_p(l_i = x) = \begin{cases} 1 & \text{if } l_i = x, \\ 0 & \text{otherwise.} \end{cases} \tag{2.11}$$

Likelihood computations are often extended to include more realistic assumptions about the way biological sequences evolve. For the DNA inference case, model accuracy can be improved by considering different nucleotide base occurrence frequencies and different rates of substitutions among structurally similar nucleotides and dissimilar nucleotides (transition–transversion ratio). Additionally, sites in DNA sequences usually do not evolve in the same way, as some nucleotides change more frequently than other ones. This among-site rate variation is modelled by assuming the scaling of branch lengths in the phylogeny to follow a gamma distribution. A detailed formulation of likelihood according to these concepts can be found in [20].



Likelihood array for l1 =

A	C	G	T
0	0	1	0

Likelihood array for l2 =

A	C	G	T
0	0	1	0

Likelihood array for l3 =

A	C	G	T
1	0	0	0

Likelihood array for v =

A	C	G	T
0.02844212	0.00098596	0.02844212	0.00098596

$L_p(v=A) = [P_{AG}(0.10)L_p(l2=G)] \times [P_{AA}(0.10)L_p(l3=A)] = 0.0314 \times 0.9058 = 0.02844212$
$L_p(v=C) = [P_{CG}(0.10)L_p(l2=G)] \times [P_{CA}(0.10)L_p(l3=A)] = 0.0314 \times 0.0314 = 0.00098596$
$L_p(v=G) = [P_{GG}(0.10)L_p(l2=G)] \times [P_{GA}(0.10)L_p(l3=A)] = 0.9058 \times 0.0314 = 0.02844212$
$L_p(v=T) = [P_{TG}(0.10)L_p(l2=G)] \times [P_{TA}(0.10)L_p(l3=A)] = 0.0314 \times 0.0314 = 0.00098596$

Likelihood array for r =

A	C	G	T
0.00083894	0.00008510	0.02420104	0.00005802

$L_p(r=A) = [P_{AA}(0.10)L_p(v=A) + P_{AC}(0.10)L_p(v=C) + P_{AG}(0.10)L_p(v=G) + P_{AT}(0.10)L_p(v=T)] \times P_{AG}(0.10)L_p(l1=G)$
$= (0.02576287 + 0.00003096 + 0.00089308 + 0.00003096) \times 0.0314 = 0.00083894$

$L_p(r=C) = [P_{CA}(0.10)L_p(v=A) + P_{CC}(0.10)L_p(v=C) + P_{CG}(0.10)L_p(v=G) + P_{CT}(0.10)L_p(v=T)] \times P_{CG}(0.10)L_p(l1=G)$
$= (0.00089308 + 0.00089308 + 0.00089308 + 0.00003096) \times 0.0314 = 0.00008510$

$L_p(r=G) = [P_{GA}(0.10)L_p(v=A) + P_{GC}(0.10)L_p(v=C) + P_{GG}(0.10)L_p(v=G) + P_{GT}(0.10)L_p(v=T)] \times P_{GG}(0.10)L_p(l1=G)$
$= (0.00089308 + 0.00003096 + 0.02576287 + 0.00003096) \times 0.9058 = 0.02420104$

$L_p(r=T) = [P_{TA}(0.10)L_p(v=A) + P_{TC}(0.10)L_p(v=C) + P_{TG}(0.10)L_p(v=G) + P_{TT}(0.10)L_p(v=T)] \times P_{TG}(0.10)L_p(l1=G)$
$= (0.00089308 + 0.00003096 + 0.00089308 + 0.00003096) \times 0.0314 = 0.00005802$

$L_i(\tau) = \pi_A L_p(r=A) + \pi_C L_p(r=C) + \pi_G L_p(r=G) + \pi_T L_p(r=T)$
$= \frac{1}{4}\ 0.00083894 + \frac{1}{4}\ 0.00008510 + \frac{1}{4}\ 0.02420104 + \frac{1}{4}\ 0.00005802 = 0.006295775$
$\text{Log-likelihood}_i(\tau) = -2.200950801297739$

Fig. 2.6 Example of phylogenetic inference under maximum likelihood: likelihood computations according to the JC69 model

A simple one-character example of likelihood computation is provided in Figure 2.6 for a phylogenetic tree with three OTUs ($l1$, $l2$, $l3$), and two HTUs (v and the root node r) and branch lengths values of 0.10. This example considers the use of the JC69 evolutionary model, which defines the probabilities of observing mutation events as described in Equations 2.3 and 2.4. Assuming that $\mu = 4/3$ and t = 0.10, the probabilities $P_{xy}(t)$ in this model are 0.9058 (for the case x=y) and 0.0314 (x≠y). Following the likelihood recursive computation approach, the partial likelihood arrays for the OTUs are defined in accordance with their character state values: ['A'=0,'C'=0,'G'=1,'T'=0] for $l1$ and $l2$ and ['A'=1,'C'=0,'G'=0,'T'=0] for $l3$. From the $l2$ and $l3$ arrays, the partial likelihood of the node v for each character is given by using Equation 2.10, resulting into a partial likelihood array = [0.02844212, 0.00098596, 0.02844212, 0.00098596]. The likelihood array for the root node is computed from the v and $l1$ arrays using the same expression: [0.00083894, 0.00008510, 0.02420104, 0.00005802]. Using these partial likelihoods and the stationary probabilities defined by the model over Equation 2.9 leads to a *L(τ,JC69)* value = 0.006295775. The likelihood scores usually reported in practice are given in terms of log-likelihoods, which avoid the calculation of products in Equation 2.9. Therefore, the resulting log-likelihood in this example is -2.2009508.

2.2.4 Exploring the Search Space

When addressing phylogenetic inference as an optimization problem, one has to deal with a phylogenetic tree search space containing a huge amount of possible candidate solutions, as it will be explained in Section 2.2.6. In this sense, the representation or encoding of phylogenies plays a key role in the design of phylogenetic methods, as it determines the efficiency and effectiveness of the search engine. The generation of new solutions must be carried out by applying accurate operators over the data structures used to represent phylogenetic topologies. Two main phylogenetics encodings are distinguished: *direct tree-based representations* and *indirect distance-based representations*.

Firstly, tree representations are commonly known as the standard way of encoding solutions in phylogenetic analyses. Considering tree-shaped data structures, the algorithm operates directly over the phylogenetic tree search space. Under this representation, new phylogenies are generated by modifying starter topologies using specific topological search operators. Two main classes of operators are usually used in a tree-based phylogenetic search engine:

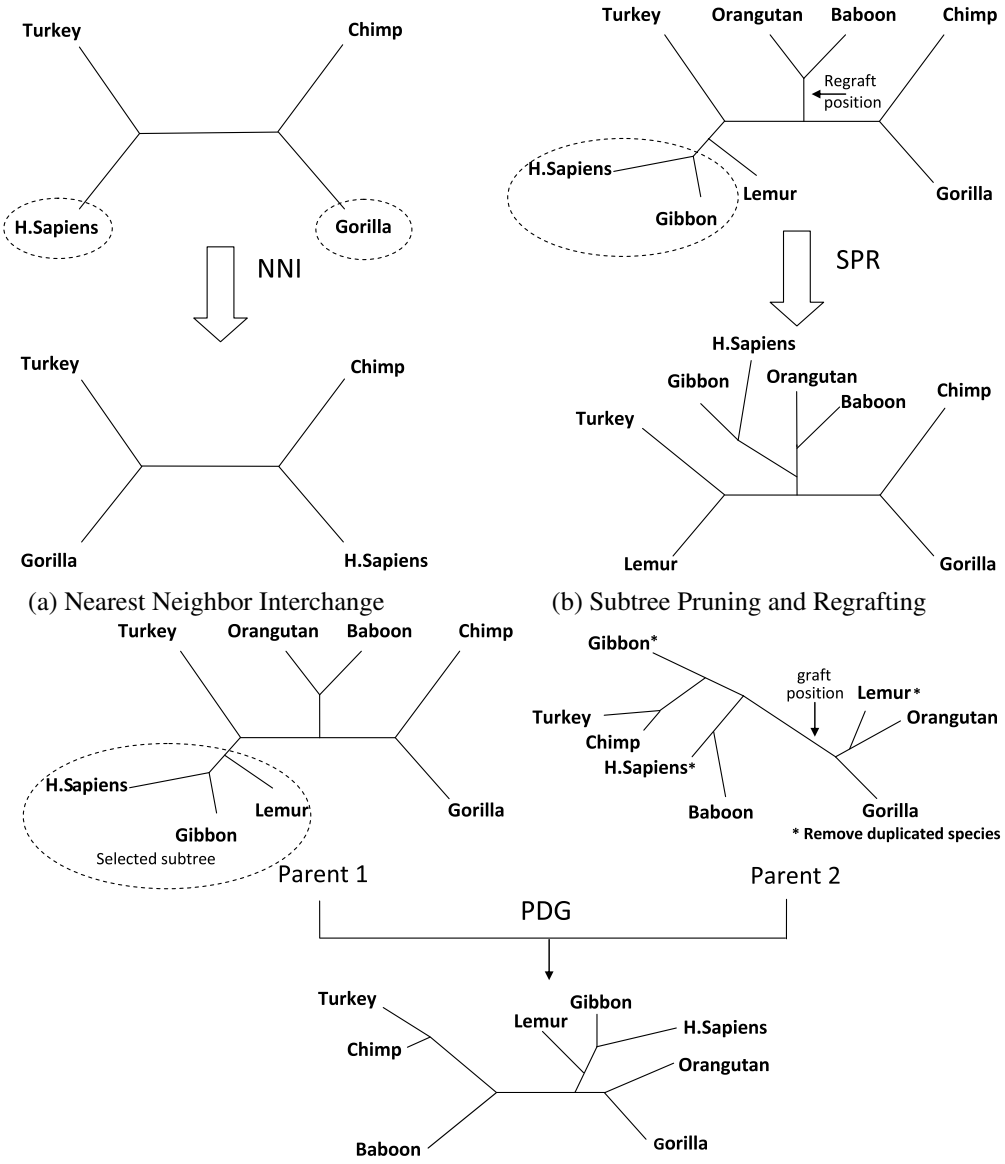
1. **Unary topological operators**. Taking as input a single phylogenetic tree, these operators alter it by applying topological rearrangements and recover a new tree with the hope of improving its quality under the used optimality criterion. Examples of unary topological operators are Nearest Neighbour Interchange (NNI) and Subtree Pruning and Regrafting (SPR) [105]. NNI takes a branch of the tree and performs a swap between the subtrees at the sides of that branch. On the other hand, SPR selects a subtree from the phylogenetic topology, removes the chosen subtree and regrafts it in a different place to generate a new tree. Figures 2.7 (a) and 2.7 (b) show examples of NNI and SPR, respectively. These two proposals can be combined to improve the effectiveness of the search, as pointed out in the Parametric Progressive Neigh-

bourhood (PPN) proposal [70]. This approach proceeds by considering a d value defined as the longest distance between all pairs of leaves, generating new neighbours by applying SPR and NNI moves at most at d steps from the pruned subtree to the regraft position.

2. **Binary topological operators**. In this case, new phylogenetic topologies are generated by combining information about evolutionary relationships from different sources, that is, from two or more input trees. The definition of binary topological operators has been closely related to the development of genetic algorithms to conduct phylogenetic searches, as they can be effectively used in the crossover step [39, 107, 114]. A representative example of binary topological operators is given by the Prune-Delete-Graft (PDG) recombination proposal [44]. This operator takes a random subtree from one of the parents and inserts it in the other parent at a randomly selected insertion point, deleting duplicated species. Figure 2.7 (c) shows a graphical example of PDG application.

A second way to encode phylogenetic trees is by means of distance matrices. Following an *NxN*-dimensional matrix-shaped representation, each entry in the matrix represents a floating-point measurement of evolutionary distance between species. In this context, two different classes of distance matrices are defined: observed matrices and inferred matrices [44]. Observed distance matrices are the ones computed from the input alignment and taken as input by distance-based phylogenetic methods, as explained in Section 2.2.2. On the other hand, inferred matrices are calculated from phylogenetic topologies by applying a tree-matrix mapping procedure which defines each entry i, j in the matrix as the length of the path between i and j in the tree. This second type of distance matrix is the one that can be used as an alternative encoding of a phylogenetic topology. A search engine which considers the use of distance matrices as phylogenetic representation operates over an auxiliary solution space, applying floating-point operators over evolutionary distances to generate new distance matrices. The main question which arises in this context is how to evaluate the quality of the resulting matrices. As observed in Equations 2.6 and 2.9, phylogenetic evaluations under parsimony and likelihood require a phylogenetic tree to perform their computations. In order to solve this issue, distance-based tree-building methods must be applied to map the solution from the auxiliary matrix space to the real phylogenetic tree space. Two main families of distance-based methods for phylogenetic topology reconstruction have been proposed in the literature:

1. **Clustering methods**. Clustering methods are a traditional approach to build phylogenetic topologies from distances. These approaches are defined on the basis of the molecular clock assumption, which considers that substitution rates in the sequences show a clocklike behaviour [59]. The results of applying these techniques are ultrametric rooted trees, which describe equidistant paths from the root node to any terminal node. Given a distance matrix m, a typical clustering method proceeds by selecting the entries in m corresponding to the species or groups of species i, j that minimize the evolutionary distance value $m[i, j]$. A partial phylogeny or cluster of species is generated by connecting i and j to a new node h which represents their common ancestor. The branch length values for (h, i) and (h, j) are then computed as $m[i, j]/2$



(a) Nearest Neighbor Interchange (b) Subtree Pruning and Regrafting

(c) Prune-Delete-Graft

Fig. 2.7 Topological operators for tree-based phylogenetic searches

and the matrix is updated with the distances to the new cluster, which replace the entries corresponding to i and j. The algorithm follows these steps until the matrix has been completely processed, returning the inferred phylogeny. Three well-known clustering methods are UPGMA [165], WPGMA [163], and complete-link clustering [99]. They mostly differ on the way they compute the distances to the new cluster:

(a) *UPGMA*. This approach assigns the updated values in m by averaging the distances according to the number of elements in the new cluster. Let *size* be the number of nodes in a partial phylogeny. The distance $D[k]$ from the cluster generated by grouping i and j to

each other group k is computed as follows:

$$D[k] = \left(\frac{i.size}{i.size + j.size}\right) m[i,k] + \left(\frac{j.size}{i.size + j.size}\right) m[j,k]. \tag{2.12}$$

(b) *WPGMA*. This proposal considers that the averaging of distances is not based on the total number of elements in the new cluster:

$$D[k] = \frac{m[i,k] + m[j,k]}{2} \tag{2.13}$$

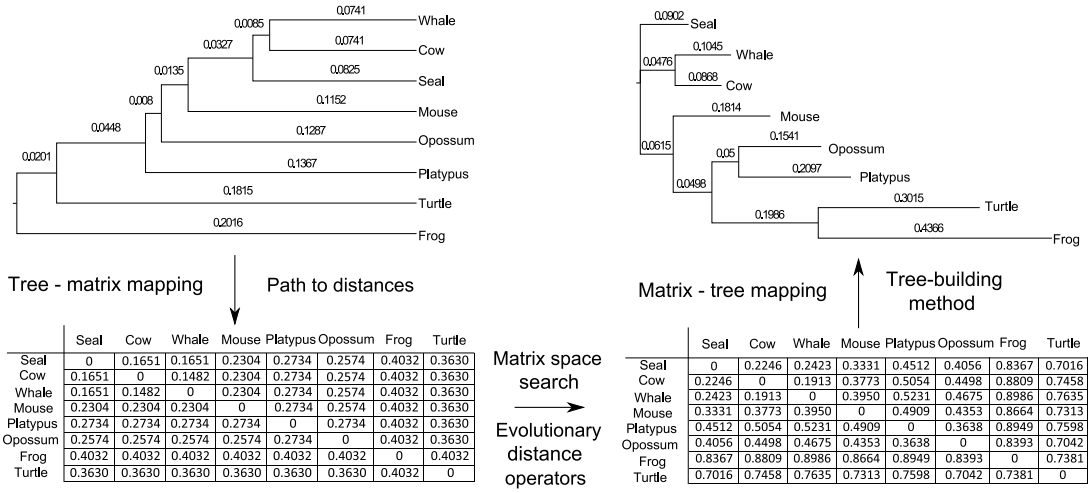
(c) *Complete-link*. In this approach, $D[k]$ refers to the largest distance from the items in the new cluster to k:

$$D[k] = \max(m[i,k], m[j,k]) \tag{2.14}$$

2. **Neighbour-joining methods**. The NJ methods differ from clustering approaches in the fact that NJ does not apply the molecular clock assumption, considering that the rates of molecular substitutions can evolve in different ways among lineages. As a result, the phylogenies generated by this kind of methods do not show the ultrametric property. A typical NJ method operates by defining, in a first step, a set of divergence values $u[i]$ for each row i in the distance matrix: $u[i] \leftarrow \sum_{j(j \neq i)}^{N} m[i,j]/(N-2)$. Those species or clusters i, j that minimize the value $m[i,j] - u[i] - u[j]$ are selected to generate a new cluster, assigning the branch length values from i and j to their common ancestor h as $h.bl_i = 1/2m[i,j] + 1/2(u[i] - u[j])$ and $h.bl_j = 1/2m[i,j] + 1/2(u[j] - u[i])$, respectively. As in a clustering approach, the matrix is updated with the distances to the new cluster, which replace the entries corresponding to i and j. These steps are repeated until the phylogeny described by the matrix has been fully recovered. A widely-used implementation of this approach is *BIONJ* [67], which improves the original NJ design by introducing a model of variances and covariances of evolutionary distances. For this purpose, a factor λ which minimizes the sampling variances of distances is computed at each iteration of the algorithm. λ is applied to compute the distances from the partial phylogeny $D[k]$ to each other group k as follows:

$$D[k] = \lambda m[i,k] + (1-\lambda) m[j,k] - \lambda h.bl_i - (1-\lambda) h.bl_j \tag{2.15}$$

Figure 2.8 shows an example of a distance-based phylogenetic search using a matrix-shaped encoding. From a starter topology, the corresponding distance matrix is computed by computing the paths between each pair of OTUs in the tree-matrix mapping step. Distance-based operators are applied afterwards to generate a new distance matrix from the original one, whose associated topology may be radically different from the starter tree. The matrix-tree mapping step is performed by applying a distance-based tree-building method, allowing the assessment of the solution according to the considered optimality criterion. Such methodology can be extended by considering that tree-



	Seal	Cow	Whale	Mouse	Platypus	Opossum	Frog	Turtle
Seal	0	0.1651	0.1651	0.2304	0.2734	0.2574	0.4032	0.3630
Cow	0.1651	0	0.1482	0.2304	0.2734	0.2574	0.4032	0.3630
Whale	0.1651	0.1482	0	0.2304	0.2734	0.2574	0.4032	0.3630
Mouse	0.2304	0.2304	0.2304	0	0.2734	0.2574	0.4032	0.3630
Platypus	0.2734	0.2734	0.2734	0.2734	0	0.2734	0.4032	0.3630
Opossum	0.2574	0.2574	0.2574	0.2574	0.2734	0	0.4032	0.3630
Frog	0.4032	0.4032	0.4032	0.4032	0.4032	0.4032	0	0.4032
Turtle	0.3630	0.3630	0.3630	0.3630	0.3630	0.3630	0.4032	0

	Seal	Cow	Whale	Mouse	Platypus	Opossum	Frog	Turtle
Seal	0	0.2246	0.2423	0.3331	0.4512	0.4056	0.8367	0.7016
Cow	0.2246	0	0.1913	0.3773	0.5054	0.4498	0.8809	0.7458
Whale	0.2423	0.1913	0	0.3950	0.5231	0.4675	0.8986	0.7635
Mouse	0.3331	0.3773	0.3950	0	0.4909	0.4353	0.8664	0.7313
Platypus	0.4512	0.5054	0.5231	0.4909	0	0.3638	0.8949	0.7598
Opossum	0.4056	0.4498	0.4675	0.4353	0.3638	0	0.8393	0.7042
Frog	0.8367	0.8809	0.8986	0.8664	0.8949	0.8393	0	0.7381
Turtle	0.7016	0.7458	0.7635	0.7313	0.7598	0.7042	0.7381	0

Fig. 2.8 Distance-based phylogenetic searches over matrix representations

based topological operators can be applied over the inferred phylogenetic topology, leading to an improvement in the search capabilities of the phylogenetic procedure.

2.2.5 Multiobjective Formulation

Section 2.2.3 has given account of the formulation of two objective functions for conducting phylogenetic analyses: parsimony and likelihood. These two optimality criteria are built upon different assumptions about the way species evolve in nature, exhibiting biological and fundamental differences [175] which give as a result conflicting outputs on real data sets [111]. In fact, the divergence shown by parsimony and likelihood analyses has generated a strong controversy among phylogeneticists for more than two decades [105]. Multiple researches [22, 111, 152] have reported conflicts in the evolutionary relationships inferred by each method, pointing out the choice of the optimality criterion as one of the most noticeable sources of incongruence in phylogenetics [146]. As a consequence, biologists are often forced to apply multiple phylogenetic software to obtain publishable results, providing different explanations for a given biological sequence alignment [32]. A way to deal with these problems consists of proposing a compromise vision of phylogenetics, addressing incongruence between parsimony and likelihood by formulating the inference process as a MOP.

Given a tree search space (decision space) SS and an objective space Z, a multiobjective approach for phylogenetics aims to find those phylogenies $\tau \in SS$ that optimize simultaneously a set of objective functions $\vec{f}(\tau) \in Z$. Identifying parsimony and likelihood as our objective functions, the multiobjective formulation of the phylogenetic inference problem can be expressed as:

$$\begin{aligned} &\text{optimize } \vec{f}(\tau) = [f_1(\tau), f_2(\tau)], \\ &\text{where } f_1(\tau) = \text{minimize } P(\tau), \\ &\qquad f_2(\tau) = \text{maximize } L(\tau, \Phi). \end{aligned} \tag{2.16}$$

Instead of searching for a single optimal solution, a multiobjective design will look for a set of high-quality Pareto solutions supported by multiple phylogenetic principles. The biological implications of this formulation can be considered from two points of view. Firstly, for those biologists interested in reporting single-criterion trees, multiobjective approaches can be applied to generate in a single run high-quality phylogenies attending to each considered objective separately. Secondly, the solutions contained in the Pareto set represent trade-offs among all the optimality criteria considered in the inference process. The usefulness of these solutions is justified in the following way. On the one hand, multiobjective phylogenetic topologies aim to address the conflicts which arise when parsimony and likelihood disagree about the best supported evolutionary histories. By considering the simultaneous optimization of both criteria, a multiobjective phylogeny is able to provide a compromise answer to the evolution of the input species, including in its topology evolutionary relationships with biological sense attending to parsimony and likelihood which are not successfully recovered by single-criterion trees separately. In addition, the analysis of the phylogenies in the Pareto set allows the expert to improve his understanding of the conflicts observed between divergent criteria and the characteristics of the solution space [77], along with having the opportunity to select the most accurate evolutionary histories by means of decision making techniques [36].

On the other hand, the information contained in the entire Pareto set can be used to introduce additional robustness to phylogenetic analyses. By combining the knowledge provided by all the topologies in the Pareto set, consensus methods [19] may be applied to generate a representative phylogenetic tree which summarizes the most supported evolutionary relationships observed in the attained Pareto solutions. Such a tree can be useful to provide more reliable conclusions from a statistical and biological point of view, as it considers information from a wide range of high-quality phylogenetic hypotheses inferred under multiobjective principles.

The biological relevance of the multiobjective approach will be assessed by experimentation in Chapter 6. Comparisons with single-criterion and multiobjective methods from the state-of-the-art will be conducted to examine the quality of the best parsimony and best likelihood phylogenetic trees, using biostatistical testing to obtain reliable conclusions. In addition, consensus trees for the different data sets analyzed in this research will be reported, combining the information contained in all the generated Pareto solutions. Finally, a case study on the solving of incongruence issues will be discussed, using for this purpose a well-known dataset of Plethodontid salamander DNA [111] where parsimony and likelihood support conflicting evolutionary relationships [32, 185].

2.2.6 Computational Complexity

Optimality criteria methods provide a methodology to guide the inference process by comparing a wide range of alternative evolutionary hypotheses. However, the reconstruction of optimal phylogenies must deal with the NP-hard nature of the problem, which has been formally demonstrated for parsimony [51] and likelihood [33]. The reasons behind this complexity are closely related to the amount of data to be processed, as the dimensions N (number of sequences) and M (number of

characters per sequence) of the input alignment characterize the size of the search space and the tree evaluation times:

- Firstly, current phylogenetic methods involve the analysis of input alignments featuring molecular characteristics from a high number of species. In such data sets, the number of alternative phylogenetic topologies increases dramatically with the dimension N of the input dataset, giving as a result an exponential growth of the phylogenetic tree search space SS in accordance with the following expression [105]:

$$|SS| = \prod_{i=1}^{n} (2i-5) = \frac{(2n-5)!}{2^{n-3}(n-3)!}. \tag{2.17}$$

  For example, if we consider $N = 20$ species, the number of possible phylogenetic topologies will be 2.22e+19. If we increase N up to 100 species, the size of the tree space grows up to 1.70e+182 possible phylogenetic trees [59]. Therefore, increasing numbers of sequences in the input alignment imply an exponential growth of the tree space size. Due to the enormous number of combinations, this issue motivates that exhaustive approaches become infeasible from a computational point of view when more than ten organisms are under study.

- Secondly, evaluation times are closely related to the length M of the input sequences [9]. Once a candidate solution has been generated, the inferred phylogeny must be evaluated to determine its quality according to the considered optimality criterion. When performing the assessment of the generated solutions, increasing values of M give rise to a linear growth of the times required to compute objective functions. For the parsimony case, the computational cost associated to Fitch's algorithm is $O(NM)$. With regard to likelihood, $2NMK$ floating-point operations are needed to calculate this objective function, where K is the number of possible states for each character in the sequences. Due to the increasing advances in DNA sequencing and the availability of large molecular sequences, the length of the input sequences has become a key factor contributing to the computational complexity of current phylogenetic analyses. In fact, the most time-consuming operations in most phylogenetic methods are enclosed in the evaluation procedures, representing percentages over 85% [8] and 90% [131] of the overall execution time in parsimony and likelihood analyses, respectively. In this sense, a multiobjective formulation of the problem requires considering the optimization of several objective functions simultaneously, thereby increasing the interest in tackling the problem by mixing multiobjective metaheuristics and parallelism.

Throughout the years, the combination of parallelism and bioinspired computing has represented a suitable paradigm to deal with this kind of computational challenges. Focusing on the bioinformatics field, the literature has reported the success of parallel bioinspired designs in a wide variety of biological problems. Some significant examples include the following:

1. Metabolic pathways identification [31].

2. Genome assembly [31, 110].
3. Knowledge discovery in microarray experiments [98].
4. Protein structure alignment and prediction [34, 49, 76, 90, 180].
5. Calibration of systems biology dynamic models [135].
6. DNA alignment and sequencing [83, 188].
7. RNA folding [154].
8. Gene network inference and pattern finding [104, 140].

Furthermore, the study of metaheuristic designs and high performance computing represents one of the most important work lines in computational phylogenetics, as it will be surveyed in the next chapter of this Thesis. The background provided in the current chapter gives account of the main issues and challenges to be considered throughout the research. In order to tackle the phylogenetic reconstruction problem, we undertake the development of multiobjective bioinspired metaheuristics supported by modern parallel computing technologies.

2.3 Summary

This chapter has detailed the main features of the problem tackled in this Thesis, defining the concept of optimization problem and its close relationship with the modelling of biological processes. Focusing on the problem of inferring evolutionary histories, the biological background behind the modelling of evolution has firstly been introduced. Afterwards, character-based and distance-based computational methods proposed for phylogenetic reconstruction have been described, defining inputs and outputs. The need to assess the quality of the inferred phylogenies from a biological perspective motivates the formulation of optimality criteria and, thereby, the definition of the inference process as an optimization problem. Two different optimality criteria are the object of study in this research: parsimony and likelihood. On the one hand, parsimony seeks for the simplest explanation to the evolution of species, in accordance with Occam's razor. On the other hand, models of nucleotide evolution and statistical measurements support the inference of phylogenetic trees under likelihood. Due to the different nature of these criteria, conflicting phylogenetic histories can be inferred for a same dataset. The scope of this Thesis lies on tackling phylogenetic reconstruction by using a multiobjective formulation of the problem, proposing a compromise view of the inference process to deal with incongruence issues. Finally, the definition of phylogenetic problem as a computationally demanding problem has been discussed, pointing out how increasing numbers of species and molecular sequence lengths contribute in a significant way to the overall complexity of phylogenetic analyses. This context justifies the application of the two computational paradigms studied in this Thesis to solve the problem: multiobjective bioinspired computing and parallelism.

Chapter 3

Related Works

Due to the difficulties which arise during the inference process, a significant amount of researches have tried to tackle the phylogenetic reconstruction problem by using metaheuristics, mostly represented by traditional genetic algorithms. In addition, algorithmic developments for computational phylogenetics have noticeably improved thanks to the advances in hardware design, making possible the processing of high-complexity data sets in reduced times through the exploitation of different levels of parallelism. This chapter gives account of the close relationship between phylogenetics, bioinspired computing, and parallelism, detailing how the literature has evolved.

3.1 Metaheuristics in Phylogenetics

The application of metaheuristics and bioinspired computing to address the inference of evolutionary trees has attracted research interest for almost twenty years [64]. This section outlines the most relevant bioinspired approaches proposed by other authors to tackle this biological problem.

Throughout the years, the growing availability of genetic data has motivated the design of different algorithmic strategies to satisfy biological processing needs. In such context, phylogenetic procedures based on optimality criteria must deal with the challenge of exploring tree search spaces which grow exponentially with the number of species. Modern data sets imply that phylogenetic methods must address the reconstruction of optimal phylogenies over very large search spaces, representing a difficult problem [40] which cannot be tackled by means of traditional exhaustive searches. This is the reason why one of the main research lines in computational phylogenetics is the development of bioinspired search mechanisms to obtain high-quality solutions in reasonable time.

The first attempt to apply bioinspired computing to phylogenetics was reported in 1995 by Matsuda [114], who published a genetic algorithm for maximum likelihood inference from amino acid sequences. This first metaheuristic design defined a crossover operator based on genetic distances and branch exchange for mutation purposes. Later on, Lewis proposed GAML [107], a genetic algorithm which significantly reduced the computational effort needed to carry out maximum likelihood analyses on nucleotide data. This algorithm addressed the reconstruction of new phylogenetic topologies

by modifying branch lengths according to a gamma distribution and performing local searches based on global topological rearrangements. In [121], Moilanen introduced PARSIGAL, a hybrid proposal for maximum parsimony which combined evolutionary algorithms and branch-swapping local search procedures. Skourikhine went a step further, and developed a genetic algorithm with self-adaptive control parameters for inferring likelihood-based phylogenies over nucleotide sequences [162]. On the other hand, Reijmers et al. suggested the application of genetic algorithms to allow phylogenetic procedures to infer satisfactory results even from bad initial topologies [141].

Significant achievements on the application of bioinspired computing to phylogenetics were due to Congdon and Greenfest [41]. These authors developed Gaphyl, a genetic algorithm for maximum parsimony phylogeny reconstruction. Initially proposed to process binary character data, Gaphyl implemented a crossover strategy which randomly selects a species from the first parent, randomly choosing a subtree that contains the specific organism and identifying the smallest subtree from the second parent which contains all the species from the first parent subtree. The crossover procedure generated new solutions by replacing the subtree from the first parent with the subtree from the second one, removing any duplicate species in the process. In addition, two mutation operators were included: a first swap operator which interchanged two leaf nodes at random, and a second operator which reorganized the nodes contained in a randomly selected subtree. The extension of this approach to nucleotide data was published by Congdon and Septor in [42], reporting a more efficient processing of the search space than the widely-used PHYLIP tools.

With the publication of high-complexity data sets, researchers turned their efforts into discussing new techniques to achieve an efficient processing of the tree search space. As a consequence, new metaheuristic developments were undertaken to carry out phylogenetic reconstruction under distance-based optimality criteria, parsimony, and likelihood. Cotta and Moscato reported in [44] direct and indirect tree encoding strategies for performing distance-based phylogenetic analyses, getting meaningful results at low computational cost. These authors also explored the implications of applying memetic algorithms to phylogenetic hierarchical clustering from distance matrices [45]. This research line was followed by Gallardo et al., who proposed the hybridization of memetic algorithms and branch-and-bound techniques to find ultrametric phylogenetic trees [66]. According to the reported results, the resulting proposal was able to outperform classical agglomerative algorithms and other efficient tree-construction methods.

Regarding modern metaheuristic designs for maximum parsimony, Ribeiro and Vianna proposed in [142] a hybrid genetic algorithm which combines local search procedures with a progressive crossover strategy based on path-relinking. This approach, known as GA+PR+LS, achieved significant solution quality while improving execution time with regard to several heuristic-based methods on different benchmark and randomly generated instances. Richer et al. published a memetic algorithm named as Hydra, whose design is based on the integration of a progressive neighborhood local search and a distance-preserving crossover operator [143]. A more recent work by Richer et al. is SAMPARS [144], an improved simulated annealing approach defined to find near-optimal parsimonious phylogenetic histories from DNA sequences. The authors of this software reported a

statistically significant improvement over the LVB software [12] on 20 well-known benchmark instances, along with small standard deviations which gave account of the robustness of the proposal.

Focusing on the likelihood criterion, Shen and Heckendorn [155] proposed a genetic algorithm for maximum likelihood reconstruction which introduced a novel discrete branch length representation to achieve a balance between topology search and edge length assignments. Poladian studied in [137] the behaviour of a genetic algorithm for maximum likelihood which discarded the traditional tree-shaped encoding in favour of a distance matrix individual representation. The main idea of this proposal laid on conducting searches over a matrix-shaped space by applying distance-based evolutionary operators, using neighbour-joining tree-building methods for genotype-phenotype mapping purposes. Poladian's proposal reported significant performance in comparison with the DNAML software from PHYLIP [58], showing the relevance of using alternative phylogenetic representations. In [200], Zwickl proposed a genetic algorithm for rapid likelihood inference, named as GARLI, whose design aimed to refine the search at topological, branch length and parameter setting levels. Later on, Helaers and Milinkovich integrated several stochastic heuristics for large phylogeny inference into the METAPIGA software [80], extending a previous research where a metapopulation genetic algorithm was proposed for maximum likelihood inference [106].

The developers of heuristic-based phylogenetic methods were aware of the significant contributions achieved by the previously described approaches. In fact, bioinspired strategies laid the foundations of the current state-of-the-art techniques in single-criterion phylogenetic reconstruction. For example, the TNT software [72], which is considered the most effective tool for maximum parsimony, includes a tree-fusing method inspired by previous crossover procedures reported in genetic algorithms. Furthermore, the design of RAxML [171], one of the reference phylogenetic methods for high performance likelihood reconstruction, is built upon the integration of different bioinspired techniques, such as simulated annealing [169].

In recent years, new developments have introduced multiobjective optimization techniques to deal with the sources of incongruence that can mislead the inference process. On the one hand, incongruence issues can appear when different sources of data provide conflicting information about evolutionary relationships, as reported by Poladian and Jermiin, who applied the first multiobjective approach to phylogenetics [138]. On the other hand, the choice of the optimality criterion represents one of the main discussion topics in phylogenetics, due to the fact that conflicting hypotheses can be inferred when using different single-criterion phylogenetic software. Coelho et al. addressed this problem by using an immune-inspired approach according to two distance-based metrics: mean-squared error and minimal evolution [35]. Cancino and Delbem suggested the application of multiobjective optimization to conduct phylogenetic analyses according to maximum parsimony and maximum likelihood [23]. For this purpose, a multiobjective genetic algorithm known as PhyloMOEA was proposed, reporting results attending to both criteria on real nucleotide data sets. PhyloMOEA is based on the design principles of NSGA-II, introducing a tree-shaped individual representation and generating new solutions by using the following operators: firstly, a crossover operator based on the PDG heuristic which generates two offspring topologies; secondly, a mutation procedure

based on the application of NNI moves to interchange the position of two pairs of neighbors along with the modification of randomly selected branch lengths by using Lewis' strategy. Their proposal was extended in [24] by considering among-site rate variation under the Hasegawa-Kishino-Yano ($HKY85+\Gamma$) evolutionary model to improve the quality of the inferred solutions.

With the release of more advanced metaheuristic designs in the multiobjective optimization area, the suitability of NSGA-II (PhyloMOEA) as the most accurate strategy to address this complex biological problem becomes an open question. This Thesis aims to provide an answer to this by examining the performance achieved by different multiobjective bioinspired metaheuristics when conducting phylogenetic analyses on real-world scenarios.

3.2 Parallelism in Phylogenetics

The second line of research we consider in this Thesis is the application of high performance computing techniques to phylogenetics. In fact, the development of parallel approaches to minimize inference times has also represented one of the most important lines in phylogenetic research throughout the years. This section aims to review some of the most important parallel designs proposed in the literature to tackle the reconstruction of phylogenetic trees.

According to Bader et al. [9], different levels of parallelism can be distinguished in phylogenetic searches. Firstly, job-level parallelism can be applied to distribute multiple independent executions of the phylogenetic algorithm on the same or different data sets. Secondly, phylogenetic search engines can be parallelized by means of coarse-grained approaches, which aim to carry out concurrent searches and evaluations according to master-worker schemes. Finally, the computation of objective functions for evaluation purposes can benefit from the use of fine-grained parallelism techniques at the loop level. Following this hierarchy, most research efforts have been focused on developing algorithmic designs to exploit coarse-grained and fine-grained parallelism levels.

Early developments addressed the parallelization of widely-used heuristic algorithms proposed in the 1990s decade, introducing message-passing techniques to exploit distributed-memory multicomputers. Following this idea, Ceron et al. published in 1998 a parallel implementation of the well-known DNAML algorithm [26] for inferring maximum likelihood trees. This parallel release was focused on the exploitation of message-passing architectures by using Parallel Virtual Machine (PVM). Two years later, Snell et al. [164] studied a master-worker proposal based on the Distributed Object Group Metacomputing Architecture (DOGMA) to perform phylogenetic inference under maximum parsimony. Schmidt et al. reported a parallel release of the TREE-PUZZLE software [151] by combining an MPI-based master-worker model with dynamic scheduling mechanisms. A parallel implementation of the heuristic FASTDNAML was proposed by Stewart et al. in [173]. In order to parallelize this tool, the authors proposed PVM and MPI implementations based on a hierarchy of master, foreman, worker, and monitor routines, which cooperate to find the highest likelihood value among a set of phylogenetic trees generated by the master process.

These first approaches had a major drawback in the fact that they included outdated search strategies for exploring the tree solution space, giving as a result inaccurate biological performance when dealing with modern data sets [9]. In addition, their high computational requirements motivated that these first parallel proposals could only be useful to carry out phylogenetic reconstruction over low-complexity instances. New researches aimed to address these issues by proposing advanced parallelization schemes for state-of-the-art phylogenetic search engines. In this sense, the research carried out by Lewis in 1998 [107] had already suggested the suitability of phylogenetic bioinspired schemes to be improved by means of parallel computing techniques, predicting the key role that these computing paradigms would play in the following years.

The developments undertaken in early 2000s confirmed Lewis' hypotheses, giving rise to different proposals who applied parallel bioinspired algorithms to phylogenetics. For example, Katoh et al. [96] reported a parallel genetic algorithm for inferring phylogenies under the maximum likelihood criterion over amino acid sequence data, performing experimentation on clusters of workstations. Brauer et al. parallelized Lewis' original design GAML [18] for DNA-based inferences by assigning each individual in the population to a different processor, in such a way that the size of the population matched the number of processors available in the machine. Other approaches proposed by Congdon and Septor [42] and Lemmon and Milinkovitch [106] applied algorithmic designs based on the management of parallel subpopulations to improve the performance of single-objective evolutionary algorithms for parsimony and likelihood reconstruction, respectively. Furthermore, the GARLI software [200] includes in its release support for parallel machines, integrating parallelism for several purposes. On the one hand, the multithreading version of GARLI aims to increase the speed of the method by using OpenMP. On the other hand, the MPI version introduces job-level parallelism to distribute a number of independent search replicates among multiple processors.

Advances on the design of multiprocessor and multicomputer parallel architectures opened the door for different developments which were focused on the application of more advanced parallel approaches to phylogenetics. Significant contributions in this field were reported through the parallelization of the RAxML software [171]. Parallel implementations of this tool for shared-memory multiprocessor systems were studied by Stamatakis and Ott in [172], evaluating the performance obtained when using MPI, POSIX threads, and OpenMP to reduce the times required by the phylogenetic likelihood function. In addition, an MPI+MPI master-worker hybrid approach for exploiting supercomputer and cluster architectures was reported in [131]. Later on, this design evolved into an MPI+POSIX threads mixed mode parallelization of RAxML [136], proposed with the aim of taking advantage of the parallel capabilities shown by modern symmetric multiprocessor clusters. For this purpose, this implementation defined a parallel design where independent tree inferences were distributed among different MPI processes, each one parallelizing the evaluation of solutions via Pthreads to minimize execution time.

In fact, the development of mixed mode parallel approaches represented a significant line of work with growing interest in the field of phylogenetics. For example, hybrid MPI+OpenMP releases of popular phylogenetic software like MrBayes [148] and IQPNNI [120] have been published in

recent years, following master-worker schemes which gave rise to improved speedups and parallel scalability in comparison to other parallel approaches [28, 119].

Maximum parsimony analyses were also conducted by combining high performance computing and algorithmic engineering. We can point out the contributions due to Moret et al. [123], who developed the GRAPPA software suite, and Bader et al. [8], who proposed a parallel exact solver for maximum parsimony known as ExactMP. Another significant research to be highlighted was due to Darriba et al., who tackled the model selection problem in statistical phylogenetics by using high performance computing techniques [48]. In addition, novel parallel designs have been introduced with the aim of performing comparative genomic analyses [127] and phylogenetic reconstruction under distance criteria from complete genome data [116].

Over the last few years, the emergence and popularity of hardware accelerators and heterogeneous computing has inspired researchers to undertake new fine-grained parallel developments for phylogenetics. More specifically, most of the heterogeneous approaches reported in the literature have as main goal the parallelization of Bayesian approaches to phylogenetic reconstruction. Suchard and Rambaut proposed novel many-core algorithms and GPU support for the BEAST tool [174], laying the foundations of the high-performance statistical phylogenetics library known as BEAGLE [7]. Other heterogeneous designs are focused on parallelizing the popular tool MrBayes [148] for Bayesian inference. In [139], Pratas et al. studied a variety of strategies to accelerate this tool by exploiting multicore CPUs, Cell/BE, and GPUs. Other MrBayes implementations for multiGPU hardware architectures and heterogeneous CPU+GPU supercomputers were proposed by Bao et al. [11] and Chai et al. [27], respectively. In 2014, Kuan et al. compared the performance obtained when accelerating MrBayes under different GPU programming platforms, proposing new parallel approaches to minimize data transfers between the CPU host and GPU devices [102]. On the other hand, GPU-based kernels and vectorization strategies were analyzed in [91] to accelerate the phylogenetic likelihood function for pure likelihood search engines, using for this purpose the recently proposed Phylogenetic Likelihood Library [63]. Regarding the hardware acceleration of the parsimony function, the most relevant approach was due to Alachiotis and Stamatakis, who analyzed FPGA implementations, OpenMP and SSE3/AVX vectorization schemes for Parsimonator [5].

Recently, novel multiobjective proposals have been applied to solve incongruence problems when different optimality criteria lead to conflicting phylogenies. As multiple time-consuming objective functions must be considered, the parallelization of these algorithmic designs becomes a key issue to carry out efficient multiobjective searches over computationally demanding data sets. This idea motivated Cancino et al. to develop MPI and MPI+OpenMP parallel schemes for the PhyloMOEA multiobjective approach [25]. In their research, these authors proposed two parallel versions of this tool. Firstly, PhyloMOEA-MPI defined a pure MPI master-worker scheme based on distributing candidate solution evaluations among MPI processes. Secondly, PhyloMOEA-Hybrid extended the previous MPI implementation by introducing OpenMP directives to parallelize objective function loops. The experimental evaluation of these parallel designs considered the analysis of four real DNA data sets over a cluster configuration composed of 16 processing cores. As a significant hint,

the authors reported that the parallel scheme leaded to a reduction of execution time from 50 to 6 hours on a well-known dataset of rbcL plastid gene data. However, noticeable overhead issues due to communications and thread synchronizations penalized the scalability of the proposal.

This context justifies the need to make further developments on the application of parallel computing to multiobjective phylogenetic inference. The increasingly important role that high performance computing has taken in the design of single-criterion phylogenetic methods motivates us to explore different parallel computing models to carry out multiobjective searches. In accordance with the most popular trends in parallel phylogenetic research, we undertake the design of different coarse-grained parallelization schemes for our multiobjective metaheuristics, with the aim of taking advantage of the parallel capabilities shown by multicore multiprocessors and clusters. This line of work is complemented by discussing fine-grained heterogeneous techniques to deal with time-consuming operations via CPU+GPU computing.

3.3 Summary

In this chapter, the most significant contributions reported by other authors to tackle the phylogenetic inference problem have been described. First of all, we focused on the relevant role played by computational intelligence and metaheuristic design in the development of search engines to conduct optimality-criteria based inferences. Starting from early researches which applied simple genetic algorithms, we have described different approaches proposed to deal with increasingly complex data sets, reporting works based on the application of memetic algorithms, simulated annealing, evolutionary algorithms hybridized with local searches, metapopulation and adaptive designs, etc. As we have shown, the literature in multiobjective approaches to phylogenetic inference requires further development, due to the fact that no comprehensive comparative studies among different multiobjective algorithms have been carried out.

The second main line of research in computational phylogenetics is the development of parallel designs to address time-consuming inferences. By reviewing the literature, we have pointed out how high performance computing techniques have supported the development of phylogenetic methods throughout the years. Although early parallel developments presented several issues due the high computational requirements associated to outdated search strategies, they laid the foundations of a wide range of researches which contributed to achieve significant advances in the phylogenetics domain. Particularly, advanced parallel designs were proposed to address the analysis of complex data sets in a successful way by exploiting multiprocessor systems, multicore clusters, and GPGPU computing. Once again, the design of parallel multiobjective phylogenetic methods has not been widely studied in the literature, so this Thesis aims to address this issue by exploring different parallel computing models and their suitability to support this novel research direction in phylogenetics.

Chapter 4

Methodology

This chapter aims to describe the methodology followed throughout the development of this Thesis. For this purpose, it begins with the explanation of useful notions about multiobjective optimization, bioinspired computing, and parallelism. First of all, further information on basic multiobjective optimization concepts is provided, introducing the main goals of a multiobjective optimizer. Afterwards, the basic bioinspired approaches which lay the foundations of the applied metaheuristics are introduced, highlighting the two main types of multiobjective algorithmic designs studied in this research. Secondly, as this Thesis is also focused on the development of parallel solutions to the problem, the examined parallel programming models and libraries are detailed, identifying the different parallelization schemes traditionally integrated into multiobjective metaheuristics. After introducing these key notions, this chapter proceeds with the description of the multiobjective, parallel, and biological performance metrics used to evaluate the applied multiobjective designs. Finally, our experimental and statistical methodology is explained, describing the biological data sets used in our experiments, our hardware setups, and other materials.

4.1 Multiobjective Optimization and Bioinspired Computing

Advances in the field of multiobjective optimization have been closely related to the development of new algorithmic approaches inspired by the behaviour of natural systems. One of the key features that explain the suitability of applying bioinspired metaheuristics, such as evolutionary algorithms and swarm intelligence designs, to this kind of optimization problems is given by the fact that these approaches often manage a set or population of solutions throughout their execution. Such population-based engines make easier the generation of multiple solutions to the problem in a single run, fitting the requirement of looking for multiple trade-offs in a MOP. In addition, bioinspired designs often represent an interesting approach to deal with very large search spaces, allowing an efficient discovery and exploitation of solutions by mimicking search strategies from nature. This section is focused on detailing general key features behind the design of bioinspired multiobjective

algorithms, introducing some key terms widely used in the field of multiobjective optimization and detailing the main lines of algorithmic designs considered in this research.

4.1.1 Some Basic Notions on Multiobjective Optimization

As formulated in Chapter 2, a MOP deals with the problem of finding those solutions $s = [s_1, s_2, ..., s_k]$ (defined by k decision variables) in a decision space SS which optimize n objective functions $\vec{f}(s) = [f_1(s), f_2(s), ..., f_n(s)]$ from an objective space $Z = \Re^n$ [36]. That is, the output of a multiobjective optimizer is given not by a single solution as in classic single-objective optimization, but a set of solutions which represent some high-quality compromises among the considered objectives. Therefore, the notion of optimality changes in a multiobjective context, as originally pointed out by Francis Ysidro Edgeworth [56] and generalized by Vilfredo Pareto [133].

One of the first questions which arise when dealing with a MOP is about the way solutions can be compared each other. In a single-objective procedure, the solution that shows the best value for the considered objective function should be preferred. In multiobjective optimization, several objective scores must be considered simultaneously. Thereby, the pairwise comparison of solutions from a multiobjective perspective is mainly conducted according to the concept of Pareto dominance [176].

Definition 4.1.1. Pareto dominance. Given two solutions s, s' from the decision space SS of a MOP involving n objectives, we say that s dominates s' (represented as $s \succ s'$) iff $\forall\ i \in [1,2,...,n]$, $f_i(s)$ is not worse than $f_i(s')$ and $\exists\ i \in [1,2,...,n]$ such that $f_i(s)$ is better than $f_i(s')$. If $s \succ s'$, then s is as good as s' in all the considered objectives and, at least, s is better in one of them. In this context, the dominant solution can be considered better than the dominated one from a multiobjective perspective.

If the second condition of the Pareto dominance relation is not verified but the solution s is still not worse than s' in all the objectives, s is said to weakly dominate s' (represented as $s \succeq s'$). The **weak Pareto dominance** is a particular case of dominance which is commonly used for multiobjective performance measurement purposes.

Definition 4.1.2. Non-dominated solution. Given a solution $s \in SS$ and a set of solutions S^*, s is said to be non-dominated with regard to S^* iff $\neg\exists\ s^* \in SS$ such that $s^* \succ s$. If we consider the case in which $S^* = SS$, then s is considered a Pareto-optimal solution.

Definition 4.1.3. Pareto optimality. A feasible solution $s \in SS$ is said to be Pareto-optimal iff $\neg\exists$ $s^* \in SS$ such that $s^* \succ s$, this is, s is non-dominated with regard to the overall decision space. If s is Pareto-optimal, then it is not possible to find a solution that improves one of the considered objectives without worsening at least other one. In a multiobjective context, there is not a single Pareto-optimal solution, but a set of Pareto-optimal solutions which define the Pareto-optimal set.

Definition 4.1.4. Pareto-optimal set. The Pareto-optimal set PS is defined as the set containing all the Pareto-optimal solutions to the problem: $PS = \{s \in SS | \neg\exists s^* \in SS, s^* \succ s\}$. These solutions represent the best possible trade-offs among the considered objectives, representing therefore the ultimate goal to be attained by a multiobjective optimizer.

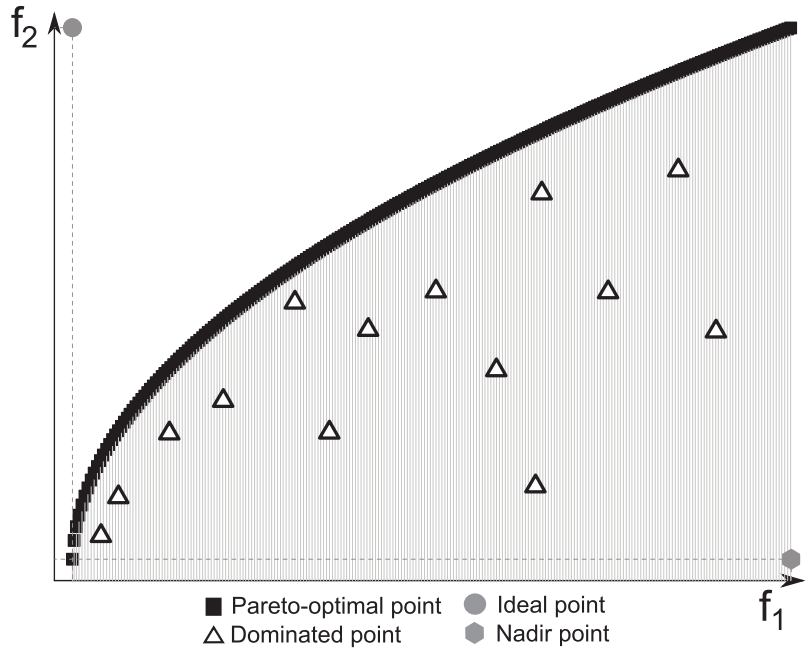


Fig. 4.1 Graphical representation of multiobjective notions

Definition 4.1.5. Pareto-optimal front. Given the Pareto-optimal set PS for a MOP, the Pareto-optimal front PF is defined as $PF = \left\{ \vec{f}(s) \in Z, s \in PS \right\}$, that is, the representation of PS in the objective space Z. Each Pareto-optimal solution in PF corresponds to a single point in the Pareto-optimal front, generating an n-dimensional curve in Z when all the solutions are plotted according to their objective function scores.

When assessing the quality of multiobjective optimizers, some special points are defined in the objective function space for reference purposes. Such reference points can represent either feasible or infeasible solutions to the problem. Examples of reference points are the ideal and nadir points.

Definition 4.1.6. Ideal point. The ideal point $Z_{ideal} \in Z$ is the point containing, for each objective function separately, the best score found throughout the Pareto-optimal front: $\forall\ i \in [1,2,...,n]$, $f_i(Z_{ideal}) = optimal f_i(s) \in PF$. The ideal point does not represent a feasible solution to the tackled problem, due to the fact that the existence of such point would imply the absence of conflict among the objectives under optimization.

Definition 4.1.7. Nadir point. The nadir point $Z_{nadir} \in Z$ is the point containing, for each objective function separately, the worst score found throughout the Pareto-optimal front: $\forall\ i \in [1,2,...,n]$, $f_i(Z_{nadir}) = worst f_i(s) \in PF$.

Figure 4.1 graphically represents the previously introduced notions, showing a Pareto-optimal front, the ideal and nadir points, and dominated points for a MOP involving the optimization of two objective functions f_1 and f_2, where f_1 is to be minimized and f_2 to be maximized.

4.1.2 Primary Goals of a Multiobjective Metaheuristic

Finding the Pareto-optimal set in real-world MOPs represents a challenging and often intractable problem. The complexity of the problem to be solved often implies that generating the Pareto-optimal set turns to be very computationally demanding, making infeasible the application of exact methods. As a result, the set of all Pareto-optimal solutions is often unknown. A multiobjective metaheuristic specifies a general-purpose template that can be used as a guiding strategy to solve specific MOPs in reasonable times. More specifically, such approaches represent approximated methods which face the NP-hard nature of a MOP by defining a search engine which leads the algorithm to high-quality solutions, although they do not guarantee to achieve the complete set of Pareto-optimal solutions. Hence, multiobjective metaheuristics aim to provide in their outcome satisfying approximations to the Pareto-optimal set, denoted as Pareto approximation sets [197].

Definition 4.1.8. Pareto approximation set. Given a set of solutions A for a MOP, A is said to be a Pareto approximation set iff any solution $s \in A$ does not weakly dominates any other solution s' in A. Ω is used to denote the set of all approximation sets to the problem.

The above definition indicated that all the solutions found by a multiobjective optimizer which are dominated by any other solution are of no interest for outputting purposes. Therefore, the outcome of a multiobjective metaheuristic will be given by a Pareto approximation set. As different metaheuristics may lead to different outcomes in accordance with the implemented search strategies, some criteria must be considered to be able to give some preference to one of the outcomes over the other ones. Traditionally, the relevance of the approximation sets generated by multiobjective approaches is assessed attending to two properties [176]:

1. **Convergence** or closeness to the Pareto-optimal front. This property states that those approximation sets which are closer to the set of Pareto-optimal solutions should be preferred, as they contain solutions with better objective scores. The convergence goal in a multiobjective optimizer ensures the generation of near-optimal Pareto solutions.

2. **Diversity** of solutions. In addition to the convergence property, it is desirable that the approximation sets contain a good spread of solutions in accordance with the shape of the Pareto-optimal front. Approximation sets including a good diversity of solutions which translates into a uniform distribution of near Pareto-optimal solutions should be preferred.

Considering these properties, the optimization goals in a multiobjective context can be defined in terms of minimizing the distance to the Pareto-optimal front and maximizing the diversity of the generated solutions [196]. Figure 4.2 shows different scenarios attending to convergence and diversity for a MOP involving two objective functions (as in Figure 1, f1 to be minimized and f2 to be maximized). The first plot given by Figure 4.2 (a) shows a near-optimal Pareto approximation set showing good convergence and diversity. Figure 4.2 (b) shows a scenario where a good spread of solutions has been attained, but the points are of limited quality in accordance with the poor closeness

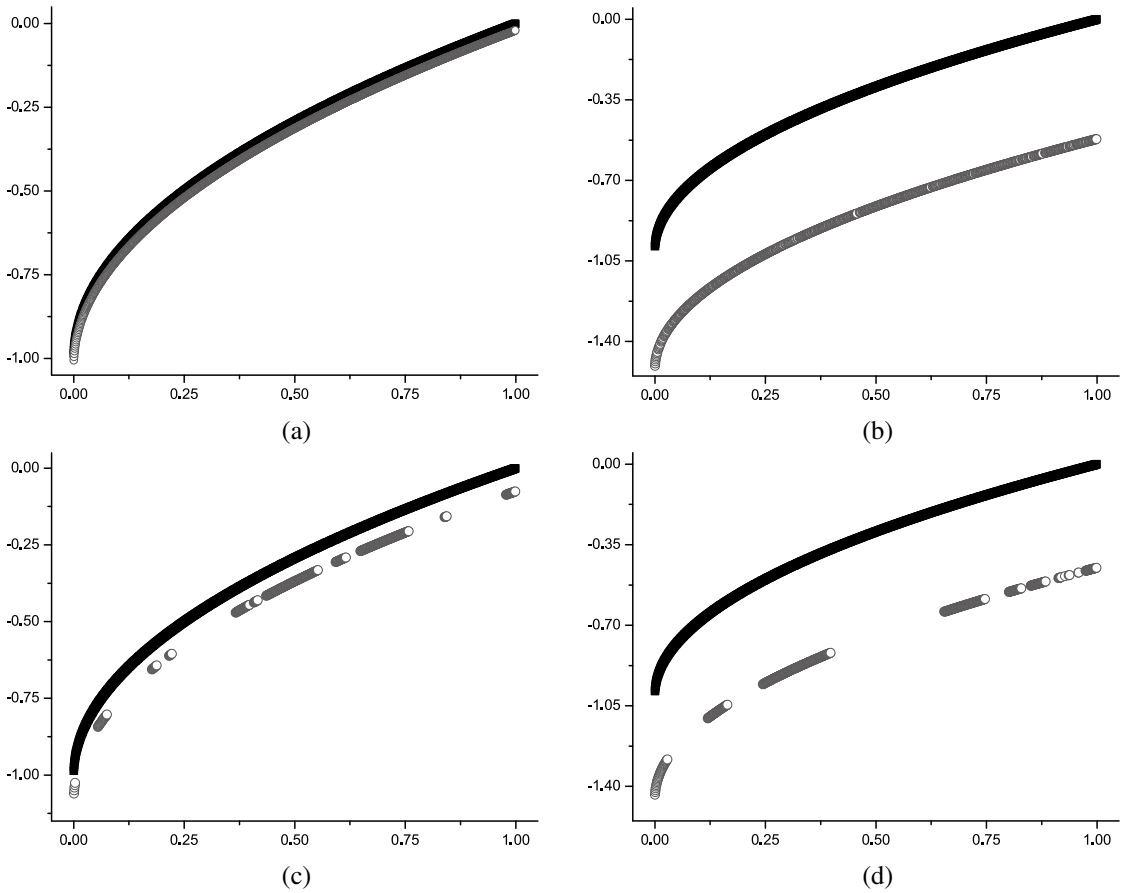


Fig. 4.2 Possible outcomes of a multiobjective optimizer, showing a) good convergence and diversity; b) good diversity but poor convergence; c) good convergence but poor diversity; d) poor convergence and diversity

shown towards the Pareto-optimal front. The third scenario in Figure 4.2 (c) shows good convergence but some regions of the objective space have not been recovered by the optimizer, losing important information on the Pareto front. The final scenario described in Figure 4.2 (d) presents an outcome with poor convergence and poor diversity.

Multiobjective Performance Measurements

One of the most important tasks to be carried out when applying multiobjective approaches is the assessment of the attained multiobjective performance. Multiple multiobjective optimizers may return as output different approximation sets, whose quality must be evaluated to decide which algorithmic approach leads to the most satisfying performance for the tackled MOP. As multiple and often very different sets of solutions must be examined, the assessment of multiobjective quality becomes more challenging than the evaluation of single-objective procedures. Fortunately, multiple researches have reported different multiobjective metrics that can be applied to conduct performance comparisons

among multiobjective approaches, providing a way to decide if an algorithm is better than other ones. Such metrics are generally known as quality indicators [197]:

Definition 4.1.9. Quality indicator. A m-ary quality indicator I refers to a function $I{:}\Omega^m \rightarrow \Re$ which maps m approximation sets (m=1 when using unary indicators, m=2 for binary indicators, and so on) to a real number measurement which can be employed to distinguish the multiobjective quality of the solutions contained in the generated sets.

According to the above definition, unary quality indicators evaluate the outcome of a single multiobjective algorithm, computing a scalar value which describes the quality of the generated approximation set in terms of convergence to the Pareto-optimal front and/or diversity. On the other hand, binary quality indicators perform direct comparisons between the approximation sets generated by two multiobjective optimizers. The computation of quality indicators often requires some additional information about the tackled problem, such as the shape of the Pareto-optimal front (which is unknown in most real-world MOPs), reference points (ideal and nadir points) and solutions sets, etc.

Different quality indicators assess the generated outcomes according to different quality criteria. Hence, these multiobjective metrics can be classified in the following way:

- **Convergence-based quality indicators**. This kind of indicators measures the quality of the evaluated approximation sets according to their closeness to the Pareto-optimal front. Therefore, these performance metrics evaluate how a multiobjective optimizer addresses the convergence goal. Examples of convergence-based indicators are the generational distance [183] and the epsilon indicator [197].

- **Diversity-based quality indicators**. These indicators evaluate the uniformity of the distribution of solutions in the objective space, in terms of dispersion and extension. They give preference to those multiobjective approaches that achieve approximation sets with a good spread of solutions. Two examples of diversity-based indicators are the spread [194] and the spacing [183] metrics.

- **Hybrid quality indicators**. Hybrid indicators were defined to provide a numerical measurement of both convergence and diversity. One of the most widely-used quality indicators in the literature, the hypervolume [14] of the objective space covered by an approximation set, belongs to this class of quality indicators.

In order to conduct reliable comparisons of multiobjective performance, it is desirable to combine information from several quality indicators, as the information they provide can complement each other. In addition, both unary and binary quality indicators should be applied, so that classifications attending to unary indicators can be combined with the pairwise comparison of approximation sets performed by binary indicators. Section 4.3.1 will provide detailed descriptions on the quality indicators considered in this research to assess multiobjective results.

4.1.3 Design of Multiobjective Metaheuristics

In order to address the hardness of a MOP, search engines in multiobjective metaheuristics must be able to carry out an efficient processing of the search space which leads to satisfying approximations to the Pareto-optimal set. For this purpose, a multiobjective design must contain an accurate fitness assignment procedure which effectively guides the algorithm towards high-quality solutions according to the convergence property. The diversity property requires that the algorithm must integrate some kind of mechanisms to promote diversity in the solutions managed by the search engine. Finally, the algorithmic designs must be able to preserve the information contained in the best solutions found throughout the execution of the algorithm in order to achieve robust performance.

The development of multiobjective search engines has been closely related to the modelling of bioinspired strategies to solve problems in a similar way as natural systems do. In this Thesis, we are interested in applying multiobjective bioinspired metaheuristics which are built on the basis of two main artificial intelligence paradigms: evolutionary computation and swarm intelligence.

Evolutionary Algorithms

As reported in Chapter 3, a wide variety of researches have put special emphasis on solving the problem of inferring evolutionary histories by using evolutionary search mechanisms. Evolutionary algorithms belong to a class of stochastic search methods which simulate natural evolutionary processes, using the Darwinian concept of the survival of the fittest. An evolutionary algorithm deals with the management of a population composed of multiple individuals, whose evolution is carried out through the application of selection, reproduction, variation, and competition mechanisms [176]. In such a context, every individual in the population encodes a candidate solution to the optimization problem to be solved. The computation of objective functions allows the algorithm to distinguish the quality of the solutions contained in the population by means of fitness assignment procedures. At each iteration of the algorithm (named as generation) the fittest individuals in the population, that is, the best candidate solutions are selected and assigned the role of parents. The information contained in these parent individuals will be used to generate new candidate solutions, denoted as offspring individuals, by means of crossover and mutation mechanisms. These offspring individuals are integrated into the population, competing with the original solutions to decide which individuals will survive. This basic scheme is repeated until a stop criterion is satisfied.

The main components in an evolutionary algorithm are listed below [176]:

1. **Representation**. The individual representation denotes how a candidate solution to the problem will be encoded. The decision on selecting an accurate individual representation represents a key question in the success of the algorithm to tackle the considered optimization problem. In evolutionary computation terminology, the term chromosome is used to refer to the data structure that encodes the solution associated to an individual. The decision variables which compose a chromosome are known as genes, while the terms alleles and locus are used to

denote the possible values of a gene and the gene position within a chromosome, respectively. Other commonly used terms related to the individual representation are genotype and phenotype. The genotype represents the full encoding of a solution while the phenotype represents the solution itself. These concepts may refer to the same (or very similar) data structures when a direct encoding of the individual is considered. On the other hand, an indirect encoding implies that genotype and phenotype refer to different structures. In order to illustrate the differences between genotype and phenotype in an indirect encoding, we refer to the example of a phylogenetic search engine which operates over distance matrices (Figure 2.8 in Chapter 2). The genotype of an individual will be its distance matrix, while the phenotype will be represented by the corresponding phylogenetic tree. In this case, a genotype-phenotype mapping is required before proceeding with the evaluation of the individual.

2. **Initialization of the population**. The initialization of the population denotes the starting point of an evolutionary algorithm. The solutions associated to the initial individuals are defined by using different initialization procedures. A common way to initialize the individuals is by setting their decision variables in a random way, generating random starter solutions which will be improved through the execution of the algorithm. Some real-world optimization problems may require more advanced initialization procedures, applying some heuristics to define a good starting point for the search.

3. **Fitness function**. Fitness procedures assess the quality of the individuals in the population attending to the objective functions to be optimized. The main goal of a fitness function is to provide a numerical measurement of quality which allows the algorithm to distinguish which individuals represent the most promising solutions to the problem. In single-criterion optimization, the fitness procedure may be as simple as computing the score of the objective function for the evaluated individual. In a multiobjective optimization context, more advanced fitness procedures which take into account multiple objective functions must be considered. For this purpose, Pareto dominance-based rankings or even quality indicators can be used for fitness assignment purposes, defining the two main lines of multiobjective designs which will be explained later in this chapter.

4. **Evolutionary operators**. These operators provide mechanisms to generate new offspring solutions from selected parent individuals. Different evolutionary operators can be distinguished:

   (a) **Selection**. The selection operator chooses the most fitting individuals to be considered as parents in the generation of offspring solutions. The main idea is to give preference to the most satisfying solutions according to the fitness function. Other worse individuals should also have some chance to be selected, due to the fact that they may contain useful information that could give rise to improved children individuals. Different strategies can be implemented for selection purposes. For example, each individual can be associated to a selection probability proportional to its fitness value, applying a roulette wheel selection

of parents. Another strategy is the tournament selection, where different individuals are compared in a tournament-like competition procedure, selecting as parents the best candidates according to their fitness values.

(b) **Crossover or recombination**. The crossover is a binary operator that takes two parent individuals chosen by means of selection operators and generates one or more offspring solutions which inherit characteristics of the two parents. Examples of crossover operators are the one-point crossover, the n-point crossover, and the uniform crossover. In a one-point crossover, a gene of the offspring individual is randomly selected as crossover point, in such a way that all the genes located before this point are inherited from the first parent and the remaining ones from the second parent. The n-point crossover generalizes this approach by considering multiple crossover points. On the other hand, each gene in the offspring solution is selected randomly from either parent in the uniform crossover.

(c) **Mutation**. The mutation is a unary operator that applies small changes to the generated offspring individuals. The idea behind mutation is to introduce some diversity in the decision variables contained in an offspring solution, avoiding big changes in the chromosome that could potentially lead to the loss of valuable features from the parents. A common way to apply a mutation is to consider a low mutation probability to modify a few randomly selected genes in the chromosome in accordance with the range of the decision variables of the problem.

5. **Integration and competition**. This component deals with the integration of the offspring individuals into the population, competing with the old individuals to survive in the next generation of the algorithm. An elitist approach will define that only the best individuals will compose the population of the next generation, although some stochastic procedures can be applied to allow the survival of worse individuals which promote diversity in the resulting population.

6. **Stop criterion**. It defines the moment in which the evolutionary algorithm must stop its execution. Some widely-used stop criteria defined in the literature are given by a maximum number of generations, maximum number of evaluations of the fitness function, execution time, etc.

Figure 4.3 shows a graphical representation of the main steps in an evolutionary search engine. These algorithms usually takes as input several parameters which require configuration, such as the population size (number of individuals which compose the population), the crossover probability (how often the crossover operator will be applied in the generation of offspring solutions), and the mutation probability (how often the mutation operator will be applied over the chromosome). The term Multiobjective Evolutionary Algorithm (MOEA) is usually used to refer to those evolutionary-based metaheuristics designed to address MOPs.

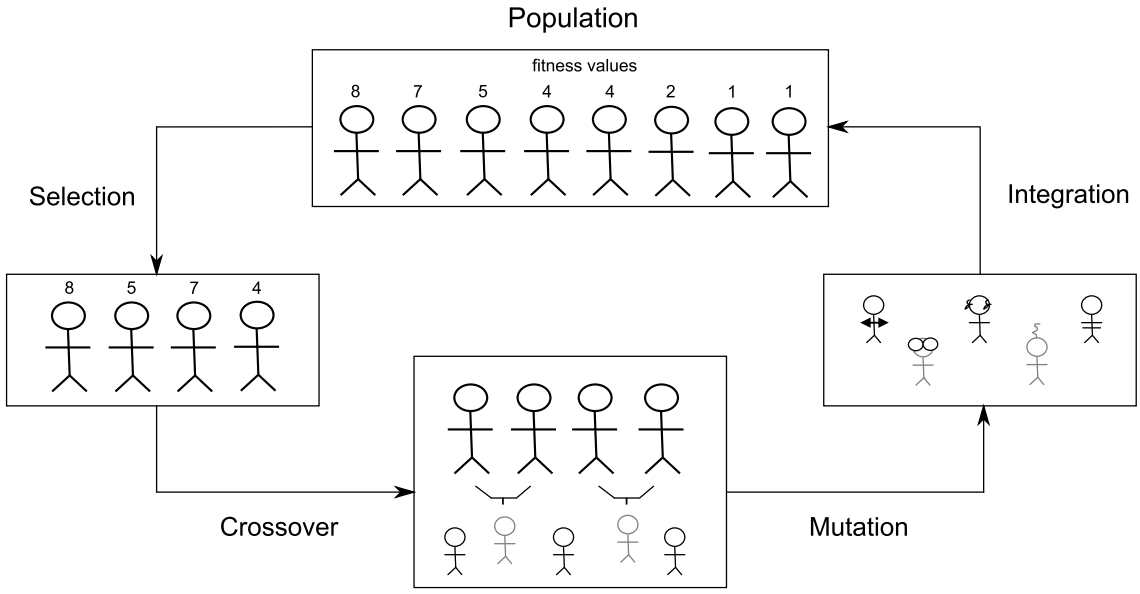


Fig. 4.3 Main steps in an evolutionary approach

Swarm Intelligence

Another line of research in the development of metaheuristics is the design of bioinspired approaches based on the collective intelligence shown by social animals in nature, such as bees, fireflies, ants, birds, and bats. Such organisms show an interesting behaviour when dealing with problems like food finding. By themselves, a single member of these animal societies do not show advanced intelligence patterns. However, task divisions and interactions between members can give rise to a coordinated behaviour which allows the whole society to address the problem effectively, achieving swarm intelligence [15]. The concept of swarm intelligence was originally introduced by Beni in 1989 [13], laying the foundations of a wide variety of researches that undertook the development of advanced search mechanisms for numerical optimization inspired by these social species.

A swarm intelligence system defines a set (swarm) of autonomous agents with well-defined roles which interact each other to achieve a common goal. A swarm can be composed of different classes of agents, defined in accordance with a *division of labour* which specifies the tasks to be performed by each agent. In such a system, the actions of the different classes of agents are governed by local rules which describe how the agents must carry out their tasks in the environment. The sharing of information among agents leads to the *self-organization* of the swarm. More in detail, self-organization refers to the appearance of a highly structured collective behaviour in a system from the interactions of its lower-level components. That is, different classes of agents perform tasks and gather useful information which is communicated to other agents through interactions, achieving collective intelligence. In this sense, swarm intelligence follows five basic principles [118]:

1. **Proximity principle**. The agents of a swarm should be able to perform simple computations related to the surrounding environment.

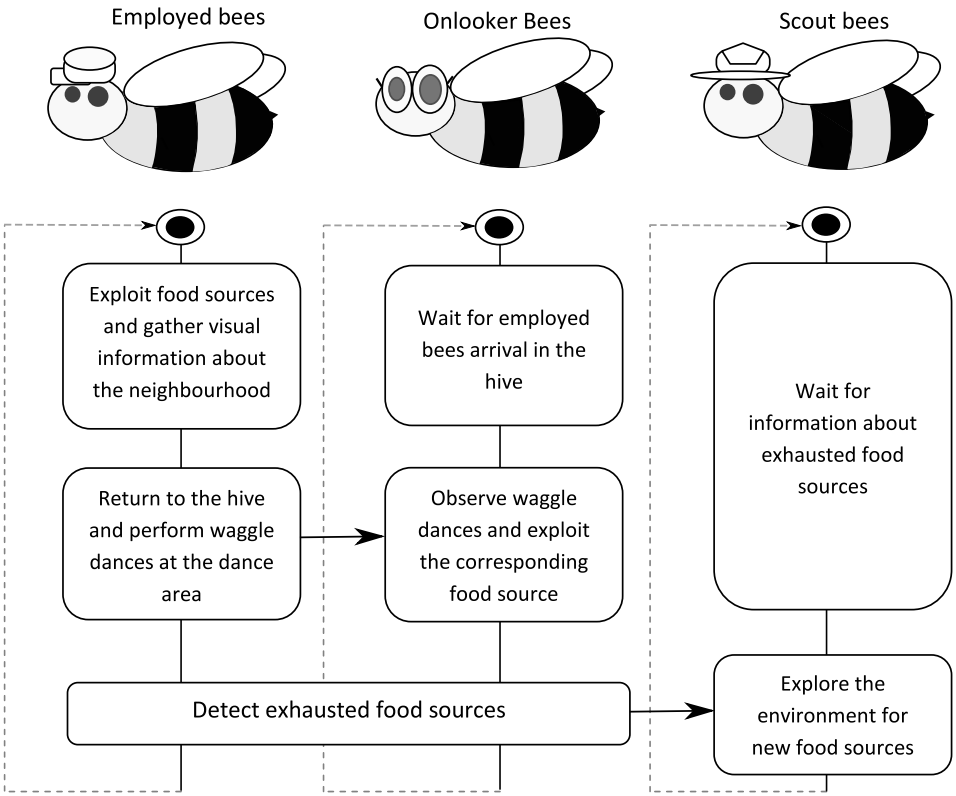


Fig. 4.4 Collective behaviour in a bee swarm, showing the main notions of labour division and self-organization via interactions

2. **Quality principle**. The whole swarm should be able to respond to quality factors in the surrounding environment.

3. **Diverse response principle**. The distribution of resources should be carried out in such a way that each agent will be protected when facing environmental fluctuations.

4. **Stability principle**. The swarm should not change its behaviour every time that fluctuations in the environment take place, as adapting to such fluctuations may not be worth in terms of energy and time costs.

5. **Adaptability principle**. The swarm should be able to change its behaviour when facing significant fluctuations in the environment.

Figure 4.4 illustrates the behaviour of a swarm intelligence system, represented by a bee swarm. When addressing the problem of finding food sources for the hive, bees are grouped into three different classes. Firstly, employed bees exploit the food sources currently known by the swarm, checking the neighbourhood for sources with higher amounts of nectar. After gathering this visual information, they return to the hive, where they interact with onlooker bees. At the dance area of the hive, employed bees perform waggle dances to share the gathered information on food sources. In ac-

cordance with the observed waggle dances, onlooker bees select the most promising food sources for exploitation purposes. When the exploitation tasks performed by employed and onlooker agents lead to exhausted food sources, scout bees are sent to check the environment for new undiscovered sources, with the aim of avoiding the starvation of the hive. This scheme shows the main principles of task division, local rules, interactions between agents, and adaptation to changes in the environment carried out by these social insects to attain collective intelligence. This behaviour models the algorithmic design of the Artificial Bee Colony algorithm [95].

In an optimization context, swarm intelligence approaches show some similarities with evolutionary algorithms: they manage a swarm (population) of agents (individuals) associated to different solutions to the problem, applying local rules and performing tasks (operators) to generate new solutions which will be improved (evolution) attending to some quality measurement (fitness) until a stop criterion is satisfied. The main difference lies on the interactions and self-organization of the agents in the swarm, which usually gives rise to advanced search engines which consider the information gathered by all the agents to guide the algorithm towards accurate solutions. In addition, swarm-based search engines show more variability in their algorithmic schemes than evolutionary approaches, as they model different patterns of natural behaviour (related to food finding, partner mating, echolocation, etc.). As a result, the input parameters for these kinds of metaheuristics strongly depend on the specific algorithmic design and the implemented operators and interaction rules.

Classification of Multiobjective Designs

Multiple multiobjective metaheuristics have been proposed in the literature. One of the most popular criteria used to classify this kind of optimizers is based on the fitness assignment scheme implemented to distinguish solution quality from a multiobjective perspective. The fitness assignment procedure is introduced to map a vector of objective scores (associated to a particular solution) to some scalar value which measures the multiobjective relevance of the evaluated solution. Attending to the implemented fitness strategy, different main lines of multiobjective algorithmic designs can be defined. In this Thesis, we are interested in studying two of the most popular classes of multiobjective algorithms [176]: dominance-based designs and indicator-based designs.

1. **Dominance-based approaches**. Dominance-based multiobjective designs conduct the search towards good Pareto set approximations by distinguishing solution quality in accordance with the Pareto dominance concept. As opposed to traditional multiobjective fitness schemes which applied objective aggregation and separate treatment of objectives, the fitness assignment procedure in these methods allows the ranking of the solutions managed by the algorithm at each generation attending to dominance. This idea was initially introduced by Goldberg in [71]. Pareto rankings establish an order among the solutions in the population, classifying them in accordance with their proximity to the Pareto-optimal front (convergence). Several examples of dominance-based fitness schemes are the dominance rank, dominance count, and the dominance depth. While the dominance rank gives the number of solutions in the population

that dominate the examined solution, the dominance count measures the number of solutions dominated by the assessed solution. In the dominance depth approach, the solutions are decomposed into several fronts F_0, F_1, F_2, ... , F_n where F_0 represents the non-dominated solutions (rank = 0), F_1 the solutions dominated only by solutions in F_0 (rank = 1), F_2 the solutions dominated by F_0 and F_1 (rank = 2), and so on. These ranking procedures are usually complemented by using density estimation measurements to refine the search and promote an uniform diversity of solutions in the approximation set, avoiding too overcrowded regions in the resulting Pareto front. Under these schemes, the optimization goal is defined in terms of minimizing the distance to the Pareto-optimal front and maximizing the diversity of the generated solutions [196]. In this Thesis, four dominance-based metaheuristics will be assessed:

- Multiobjective Artificial Bee Colony Algorithm (MOABC), a dominance-based multiobjective adaptation of the Artificial Bee Colony (ABC) [95].
- Multiobjective Firefly Algorithm (MO-FA), a dominance-based multiobjective adaptation of the Firefly Algorithm (FA) [190].
- Fast Non-Dominated Sorting Genetic Algorithm II (NSGA-II), a reference dominance-based MOEA proposed by Deb et al. [52].
- Strength Pareto Evolutionary Algorithm 2 (SPEA2), a reference dominance-based MOEA proposed by Zitzler et al. [196].

2. **Indicator-based approaches**. An increasingly popular trend in the development of multiobjective optimizers is the use of performance quality indicators to drive the search towards the Pareto-optimal front. As pointed out in Section 4.1.2, a quality indicator is a function which maps one or more approximation sets into a real number that measures the quality of the contained solutions in terms of convergence to the Pareto-optimal front and/or diversity. Indicator-based proposals go a step further in the development of multiobjective designs by defining fitness assignment and selection mechanisms on the basis of these quality indicators. In this way, the optimization goal is to generate a set of solutions that optimize the considered metric. The integration of quality indicator computations into multiobjective searches was suggested by Zitzler and Künzli [195], who proposed a general framework for indicator-based optimization. By applying quality indicators for fitness assignment purposes, decision maker preferences can be easily incorporated into the optimization process, while no diversity maintenance is required as it is implicitly incorporated by the quality indicator. On the other hand, the main disadvantage of indicator-based algorithms is given by the increased execution times required by the fitness procedure, which could be specially remarkable when tackling many-objective optimization problems (MOPs involving four or more objectives). In this Thesis, two indicator-based metaheuristics will be considered:

   - Indicator-Based Multiobjective Bat Algorithm (IMOBA), an indicator-based multiobjective adaptation of the Bat Algorithm [191].

- Indicator-Based Evolutionary Algorithm (IBEA), a reference indicator-based MOEA proposed by Zitzler and Künzli [195].

4.2 Parallel Computing Paradigms

A multiobjective algorithm involves a well-defined set of tasks to be performed to address a MOP. In this sense, some tasks must run serially one after another due to data dependencies, while others can be executed concurrently in parallel. Parallel computing paradigms provide useful techniques to support the solving of computationally demanding problems by exploiting the parallel capabilities of current hardware architectures [53]. In this section, the main parallel programming paradigms studied in this Thesis are introduced.

Current computers do not follow the idea of a single-core single-processor anymore. They now include multiple processing units which collaborate to reduce the execution time of applications suitable to be parallelized (i.e. at the instruction, loop, function, task or program levels). Two main types of streams are involved in the computations performed by a processing unit: instructions and data. On the one hand, the instruction stream comprises the sequence of operations (instructions) to be carried out by the processing unit. On the other hand, the data stream defines the information (data) to be processed by the instruction stream, flowing between the memory system and the processing unit. Attending to the number of instruction and data streams managed by a system (single or multiple), the well-known Flynn's taxonomy defines four main classes of computer architectures [81]:

- **Single Instruction Stream, Single Data Stream (SISD)**. In this category, a single instruction stream operates over a single data stream, that is, only one instruction is executed at a given time over a single data element. Single-processor von Neumann computers fall into this class.

- **Single Instruction Stream, Multiple Data Stream (SIMD)**. This category defines a single instruction stream which operates over multiple data streams, in such a way that all the processing units execute the same instruction on different pieces of data. Traditional pipeline vector processors and processor array computers are included in this category. In addition, modern processors define in their instruction sets SIMD extensions to enable the application of the same instruction on multiple data objects. GPUs also are built upon SIMD principles.

- **Multiple Instruction Stream, Single Data Stream (MISD)**. In this category, multiple instruction streams operate over the same data stream. An example of MISD consists of a same stream of data which flows through a linear array of processors running different instructions over the data. Although this class is mostly theoretical, some authors consider that systolic-array computers and pipelined machines may be included into this category.

- **Multiple Instruction Stream, Multiple Data Stream (MIMD)**. This category includes those architectures where multiple instruction streams operate over multiple data streams, that is, a group of processing units independently execute different instructions over different data.

Hence, each processor runs its own instructions on its own data. Current multicore multiprocessors and multicomputers are representative examples of MIMD architectures.

In accordance with Flynn's taxonomy, two main kinds of parallelism can be exploited [129]: data parallelism and task parallelism. Data parallelism consists of performing in a concurrent way the same function or instruction over different data in a SIMD/SPMD (Single Program, Multiple Data) programming style. Regarding task or functional parallelism, it is based on the idea of executing in parallel the same or different functions, blocks, and/or instructions which compose the application. In other words, the tasks which involve the execution of the algorithm are distributed among multiple processing units.

In this research, we are interested in examining three main parallel architectures: shared-memory multicore multiprocessors, distributed-memory multicomputers (and their extension to hybrid shared-distributed memory systems found in modern multicore clusters), and the increasingly popular heterogeneous systems involving multicore CPUs and hardware accelerators.

4.2.1 Shared-Memory Systems

A shared-memory system [81] is composed of several processors which share the same memory address space. In this kind of architectures, the processors communicate by reading and writing locations in a global memory equally accessible by all of them. When a processor writes data on the global memory, the written information will be available for the remaining processors for operational purposes. Therefore, the global shared memory represents the place where all processor coordination and synchronization takes place. Figure 4.5 represents a shared-memory system enclosing n independent processors with local cache memories, a set of m shared memory modules and I/O resources, and an interconnection network (usually buses or crossbar switches, although they may be replaced by more sophisticated networks in large-scale multiprocessor systems). This example represents a symmetric multiprocessor system (SMP), in which multiple homogeneous processors share system resources that can be accessed equally, showing balanced memory access times. Multicore processors are also included into this kind of shared-memory systems, as they can be viewed as a multiprocessor where all its 'processors' (cores) are located at the same chip and all the cores share the main memory. Furthermore, multicore multiprocessors systems (each processor containing a multicore chip) are also widely spread in modern hardware platforms.

A general classification of shared-memory multiprocessors distinguishes three main classes: uniform memory access (UMA), non-uniform memory access (NUMA), and cache-only memory architecture (COMA). In a UMA system, all processors show equal memory access time to any memory location, showing equal opportunities to perform read/write operations to memory. On the other hand, NUMA systems are based on the inclusion of local memories for each processor, which are also part of the shared memory address space. Although any process is able to access any memory location independently of belonging to the global or the local memories, the access times change in accordance with the distance of the required memory module to the processor. Accesses to local

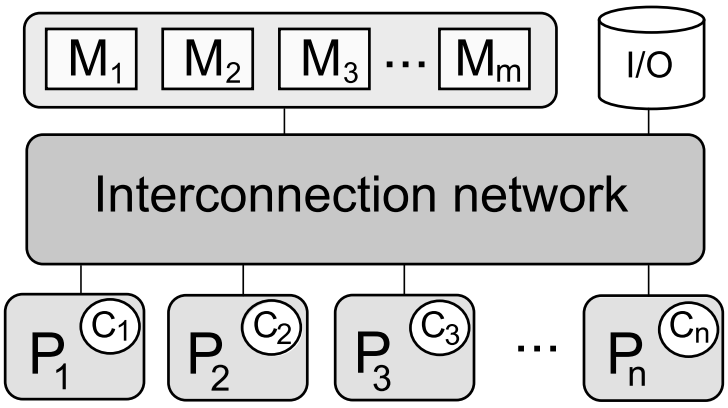


Fig. 4.5 Scheme of a shared-memory symmetric multiprocessor system

memory owned by the processor show lower times, followed by the accesses to the global memory. The highest memory access times are related to accesses to local memories from other processors. Regarding the COMA architecture, the shared memory is exclusively composed of caches. Accesses to remote caches are managed by a cache directory, while migration and replication policies must be implemented to allow memory sharing among processors.

Shared-memory multiprocessors systems show several design issues that must be considered. First of all, the possibility of multiple processors requesting conflicting read/write and write/write operations on the same memory location can endanger data integrity. Synchronization techniques must be implemented to guarantee that memory operations from multiple processors do not lead to uncertain outcomes. Semaphores, locks, barriers, and other methods represent useful mechanisms to deal with the data integrity issue when programming shared-memory systems. In addition, performance degradation might take place when the processing units are trying to make simultaneous accesses to the global shared memory, due to increased memory latency. Another common issue in these systems is cache coherence. Accesses to frequently used data are improved by introducing cache memories for each processor, which store copies of these commonly accessed data. Different caches can contain different copies of the data, leading to two possible scenarios. If the copies in the caches have the same value, the copies are said to be coherent. Otherwise, if one or several processors contain different values in their caches (i.e. after a writing operation), the copy becomes incoherent. Also, if one processor changes the contents of the memory address where the data copied in the caches was originally located, the copies will be out of date. Cache update policies and coherency protocols are included to address these problems.

4.2.2 Programming Shared-Memory Systems: OpenMP

OpenMP (Open Multi-Processing) [29] represents the industry standard for parallel programming on shared-memory systems. OpenMP is an application programming interface (API) designed to ease the development of portable parallel programs that execute on SMP architectures, multicore

machines, and other multithreaded multiprocessor systems. This API, defined for the Fortran, C, and C++ programming languages, allows the parallelization of an existing serial code in an incremental approach, identifying which portions of the program are suitable to be parallelized and enabling the sharing of work among multiple OpenMP execution threads which run on different processing units in a shared-memory framework. The OpenMP Architecture Review Board (http://www.openmp.org) is the organism which oversees the specification of the standard and approves the new features to be included in future revisions of the API.

An OpenMP application defines a block-structured approach for carrying out parallel computations under a fork/join execution model [53]. In this model, the main execution thread splits into a number of parallel threads (fork) which conduct the processing of parallel tasks. Upon termination of the defined parallel tasks, the sequential execution is resumed (join), releasing the resources used for parallel thread management purposes. A fork/join block is known in OpenMP as a parallel region, which is specified by using a compiler directive named as *#pragma omp parallel*. A parallel region defines blocks of code to be replicated across a thread pool. In a parallel region, the tasks showing parallelism opportunities can be distributed among the threads defined in the thread pool. Worksharing directives are specified in OpenMP for this purpose, enabling the distribution of tasks, iterations of a loop, etc. among multiple execution threads. Compiler directives also allow the definition of sequential sections of code to be performed by a single thread, critical sections, synchronization mechanisms, etc. Therefore, OpenMP specifies a high abstraction parallelization framework, in which the programmer manually inserts directives to guide an OpenMP-compatible compiler in the generation of shared-memory parallel applications.

In summary, OpenMP is built upon the following key components:

- **Compiler directives**. They allow the programmer to instruct the compiler on how to parallelize the code, pointing out where thread creation, data structures management, workload sharing, and thread synchronization take place. Directive clauses are introduced to define additional information, such as data scope definition (shared, private), scheduling policies in a worksharing loop directive (static, dynamic, guided), reduction operations (reduction), number of threads to be considered (num_threads), and so on. Some examples of compiler directives involve the following tasks:
  - Parallel region creation (*#pragma omp parallel*).
  - Worksharing (*#pragma omp for*, *#pragma omp sections*).
  - Serial block definition (*#pragma omp single*).
  - Critical sections (*#pragma omp critical*), atomic operations (*#pragma omp atomic*), and barrier specification (*#pragma omp barrier*).
  - Memory consistence (*#pragma omp flush*).
  - Thread creation and worksharing combination (*#pragma omp parallel for*)

- **Runtime library routines**. Specified in the header *omp.h*, these routines aim to manage and control the parallel execution environment, providing information about threads and processors, execution time, locks, etc.
- **Environment variables**. These flags define the behaviour of an OpenMP application, in terms of number of threads launched in a parallel region, scheduling policies, thread-processor affinity, nested parallelism, thread limit imposition, etc.

Algorithm 1 Vector addition with OpenMP

```
1:  #include <omp.h>
2:  #include <stdio.h>
3:  #define MAX_THREADS 8
4:  void vector_addition (int* A, int* B, int* C, int length)
5:  {
6:     int i;
7:     #pragma omp parallel num_threads(MAX_THREADS) shared(A,B,C,length)
8:     {
9:        printf("Thread identifier: %d of %d", omp_get_thread_num(), omp_get_num_threads());
10:       #pragma omp for private(i) schedule(static)
11:       for(i = 0; i<length; i++)
12:          C[i]=A[i]+B[i];
13:    }
14: }
```

Algorithm 1 shows an example of a parallel vector addition written with OpenMP. In order to enable the OpenMP API, it is required the inclusion of the header *omp.h* (line 1 in Algorithm 1). This code involves the processing of two arrays A and B, storing the results of the addition A+B into C. This code is suitable to be parallelized, as each thread can compute additions on different elements of the arrays concurrently. The initialization of the thread pool is carried out by using the *#pragma omp parallel* directive (line 7), using directive clauses to specify the number of threads (*num_threads(MAX_THREADS)*) and the sharing of the array pointers and length (*shared(A,B,C,length)*). Inside the parallel region, each OpenMP thread prints its identifier and the number of threads launched in the parallel region (line 9) by using the library routines *omp_get_thread_num()* and *omp_get_num_threads()*, respectively. Afterwards the worksharing directive *#pragma omp for* (line 10) applies over the immediately following for loop (line 11), in order to distribute different iterations among the defined execution threads. As each thread operates over different elements, the iterator 'i' must be declared as *private* in order to make private copies of this variable for each thread. This loop is performed under a *static* scheduling policy, which assigns (by default) length/MAX_THREADS iterations to each thread. When dealing with loops which show load imbalance, other scheduling policies, such as guided (where iterations are assigned in exponentially decreasing blocks) and dynamic (where iterations are dynamically assigned as the threads request them), are usually applied to improve performance. Over the defined blocks, the threads perform the

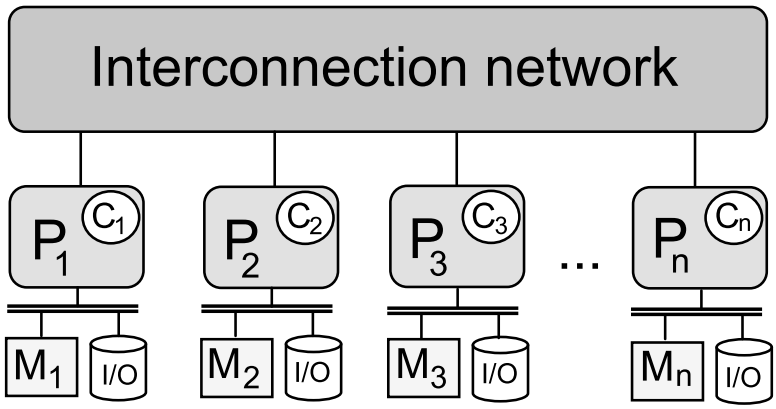


Fig. 4.6 Scheme of a distributed-memory multicomputer system

addition of the corresponding array entries in parallel (line 12). At the end of a worksharing directive, an implicit barrier is located to synchronize the threads involved in the loop processing. After that, the parallel region finishes (join) and the procedure returns in C the computed results. Further information on OpenMP programming can be found in [29].

4.2.3 Distributed-Memory Systems

A distributed-memory system [81] comprises a set of processors which do not share a global memory address space. In general terms, this kind of systems are composed of multiple computing nodes, each one containing a processor and its local memory, interconnected by a communication network which can be a commercial interconnection or architecture-specific network. The idea is to define a scalable architecture which offers flexibility in managing a large number of processing units. Figure 4.6 shows an example of distributed-memory multicomputer composed of *n* nodes. There is not a globally shared memory system and each processor only has direct access to its own local memory address space. If one particular processor requires data stored in the local memory of a different processor, a communication will take place in which messages flow through the interconnection network to request the data and retrieve a copy from the node where it is located. More specifically, the lack of global memory motivates the need to apply message-passing techniques to move data from one node to another.

In a message-passing paradigm, a message represents the logical unit for inter-node communication. Distributed-memory architectures define a set of primitives that allow nodes to communicate with each other by means of messages. The most important primitives are *send* and *receive*. While the *send* primitive sends a memory buffer through the interconnection network to the node that requested the data, the *receive* primitive allows that node to accept the message from the source node and store the data it contains into a local memory buffer. In this way, collaborations between nodes take place by interchanging messages, matching *send* requests with *receive* requests to communicate commands, tasks, results, and other data.

The performance of a distributed-memory system is closely related to the characteristics of the interconnection network, such as link bandwidth (number of bits that can be transmitted per unit of time) and network latency (the time to complete a message transfer through the network). Communication times are usually higher than the times required by memory transfers in a globally shared memory space, depending also on the size of the communicated data. Hence, communication-demanding applications may show significant overhead due to these times, requiring fast communication networks to address the impact on performance. The second issue to be considered is the synchronization among processes. It is usual to find situations where a process requires some data from a source process before going on with the processing of additional tasks. In a blocking communication pattern, if the source is not available, the processor will remain idle waiting for the requested data. The synchronization times spent until the handshake between both processors happens (including the waiting time) also contribute to the overall overhead of the application running on a distributed-memory system. Non-blocking communications can be applied to allow the process to perform (if possible) other computations which do not need the data involved in the message passing.

The most representative example of distributed-memory system is the cluster of computers. A cluster comprises multiple stand-alone computers connected by a low-latency network, which are able to operate locally over their own memories and resources and communicate with other via messages. Each node in a cluster can contain one (pure distributed memory organization) or more processors, i.e. in a SMP configuration, under a hybrid shared-distributed memory organization (Section 4.2.5 will provide more details on hybrid systems). Moreover, when the number of nodes is lower than the number of processors available within a node, the system is said to be a constellation. Fast commercial LANs or SANs with high bandwidth and low latency are usually chosen as interconnection networks. In addition, a cluster is said to be homogeneous if all nodes contained in the cluster have the same architecture features. Otherwise, we are dealing with a heterogeneous cluster.

4.2.4 Programming Distributed-Memory Systems: MPI

MPI (Message Passing Interface) [74] represents the *de facto* standard for parallel programming on message-passing distributed-memory systems. MPI specifies a portable interface for the development of message-passing programs in Fortran, C, and C++. The idea is to provide a library of routines for writing portable parallel applications under MIMD, SPMD (Single Program Multiple Data), and master-worker structure patterns, providing practicality, efficiency, and flexibility to the programmer. MPI is maintained by an open group of developers known as MPI Forum (http://www.mpi-forum.org), which defined the standard according to the following goals: design of an application programming interface under a distributed-memory framework; allow efficient communication among processes which can be run in heterogeneous environments; allow accurate integration with Fortran, C, and C++, specifying language-independent semantics; failure tolerance, as the programmer should not worry about communication failure (managed by the underlying communication system); and definition of an interface suitable to be implemented on a wide variety of platforms.

An MPI application involves a set of parallel processes which communicate each other to carry out tasks. These processes are organized in terms of groups, defining for each process a unique rank or identifier which ranges from 0 to *nproc-1* within the group, where *nproc* is the number of MPI processes which have been launched. These ranks are used to identify each process throughout the execution of the program, especially during the message-passing communications which are managed by an object known as communicator. By default, the MPI_COMM_WORLD universal communicator conducts the message-passing for all processes, although intra and inter-communicators are also available to establish hierarchies of processes. In MPI, the workload partitioning and the tasks to be performed by the processes must be explicitly defined by the developer, taking into account the characteristics of the hardware platform where the MPI application will run. In a multicomputer system, each process runs in a different processor, with direct access to its own local memory and indirect access to the data handled by other processes via message-passing. As the processes do not share a common memory address space, communications take place when one of the processes requires data from the local memory space of a different process, flowing a copy of the data from the sender memory to the receiver memory through the interconnection network. This point-to-point communication is implemented in MPI by using the functions *MPI_Send (message, destination)* and *MPI_Recv (message, source)*. The *MPI_Send* function specifies the operation of sending a copy of the data 'message' to the process with identifier 'destination'. On the other hand, *MPI_Recv* refers to the storage of the 'message' received from a process with identifier 'source'. Regarding the availability of different types of communications, the MPI standard specifies point-to-point, collective, and one-sided communications, with blocking and non-blocking synchronization models.

The six basic functions which must be considered when developing an MPI application are:

- *MPI_Init (int *argc, char** *argv)*: initialize the MPI process with 'argc' input arguments given by 'argv'.
- *MPI_Comm_rank (MPI_Comm comm, int *rank)*: provides the identifier of the process or 'rank' within the group managed by the specified communicator 'comm'.
- *MPI_Comm_size (MPI_Comm comm, int *size)*: provides the number of total launched processes or 'size' of the group managed by the specified communicator 'comm'.
- *MPI_Send (void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)*: sends a message containing 'count' data elements of type 'datatype' from the memory address specified by 'buf' to the process 'dest', with label 'tag', by using the communicator 'comm'.
- *MPI_Recv (void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)*: stores in the memory address 'buf' 'count' data elements of type 'datatype' received from a process 'source', using a message with label 'tag' handled by the communicator 'comm'. The 'status' object collects information about the communication.
- *MPI_Finalize (void)*: finalizes the MPI process, releasing resources.

Besides these functions, MPI specifies more than 440 routines which implement point-to-point communications (synchronous, asynchronous, buffered, ready, and standard models), collective operations (broadcast, gather, scatter, barrier), process groups and communication contexts management, process topologies, custom datatypes, dynamic process creation, parallel I/O, profiling interface, etc.

Algorithm 2 Vector addition with MPI

```
 1: #include <mpi.h>
 2: #include <stdio.h>
 3: #define MASTER_ID 0
 4: #define TAG 1
 5: void vector_addition (int* A, int* B, int* C, int length)
 6: {
 7:     int offset, position, i;
 8:     int myrank, size;
 9:     MPI_Status status;
10:     /* Initialization of the MPI processes */
11:     MPI_Init (NULL, NULL);
12:     MPI_Comm_rank (MPI_COMM_WORLD, &myrank);
13:     MPI_Comm_size (MPI_COMM_WORLD, &size);
14:     /* Message-passing: sending tasks */
15:     if(myrank==0){
16:         initializeInputArrays (A, B, length);
17:         /* Sending blocks of length/size elements to each MPI process */
18:         for(i=1; i<size; i++){
19:             offset = i*(length/size);
20:             MPI_Send(&A[offset], length/size, MPI_INT, i, TAG, MPI_COMM_WORLD);
21:             MPI_Send(&B[offset], length/size, MPI_INT, i, TAG, MPI_COMM_WORLD);
22:         }
23:     }
24:     else{
25:         offset = myrank*(length/size);
26:         MPI_Recv(&A[offset], length/size, MPI_INT, MASTER_ID, TAG, MPI_COMM_WORLD, &status);
27:         MPI_Recv(&B[offset], length/size, MPI_INT, MASTER_ID, TAG, MPI_COMM_WORLD, &status);
28:     }
29:     /* Performing vector addition over the assigned block */
30:     position = myrank*(length/size);
31:     for(i=position; i<position+length/size; i++)
32:         C[i] = A[i] + B[i];
33:     /* Message passing: results recovery */
34:     if(myrank==0){
35:         for(i=1; i<size; i++){
36:             offset = i*(length/size);
37:             MPI_Recv(&C[offset], length/size, MPI_INT, i, TAG, MPI_COMM_WORLD, &status);
38:         }
39:     }
40:     else
41:         MPI_Send(&C[offset], length/size, MPI_INT, MASTER_ID, TAG, MPI_COMM_WORLD);
42:     /* Finishing MPI */
43:     MPI_Finalize();
44: }
```

Algorithm 2 shows the MPI implementation of the parallel vector addition example. The MPI functions require the inclusion of the header *mpi.h* (line 1 in Algorithm 2). The first steps in the application involve the initialization of the MPI processes, obtaining identifiers and the number of processes (lines 10-13). The next step to perform is the explicit partitioning of the arrays to allow each process to carry out the addition over different blocks. The process with rank #0 contains in its local memory the complete input arrays A and B, and, after the initialization, proceeds with the

sending of blocks of length/size elements to each other process by using *MPI_Send* (lines 15-23). The remaining processes wait for the availability of the sender process #0, receiving with *MPI_Recv* the assigned blocks of data upon completion of the communication (lines 24-28). Afterwards, the MPI processes carry out in parallel the vector addition over the assigned blocks. The obtained results are sent to the process #0, which stores in the corresponding positions of the array C the received elements (lines 33-41). The application finishes with the call to the *MPI_Finalize* function (line 43). Further details on MPI programming can be found in [74].

4.2.5 Hybrid Shared-Distributed Memory Systems

As pointed out in Section 4.2.3, the popularity of SMP machines and multicore processors have led to the combination of shared and distributed memory organizations in current cluster architectures. Multicore clusters are composed of a set of nodes linked by an interconnection network where each node contains a multicore (multi)processor under a hybrid shared-distributed memory model. The main goal behind this approach is to exploit the strength of both models [53]: the scalability of distributed-memory systems and the efficiency, memory savings and easy programming of shared-memory systems. In order to take advantage of the parallel capabilities of these systems, a programming model based on the mixture of multithreading and message passing is required. The key idea is to achieve maximum efficiency by defining an accurate balance between computational and communication loads at the intra-node and inter-node levels.

When programming hybrid systems, the developer can follow two different lines. Firstly, the application may consider multiple single-thread processes for each node, using the same parallel program written for a pure distributed-memory system. The main advantage is that no additional programming effort is required, since the same program is running with a increased number of processes. However, the communication of such processes still relies on message passing even at the intra-node level. Although intra-node communications might be optimized in terms of communication latency and bandwidth, this approach could potentially lead to worst performance than a shared-memory management of intra-node resources (as other overhead sources associated to message passing are still involved, i.e. message setups costs and buffer management). Secondly, the application may involve one (or several) multithreaded process(es) per node. The intra-node communication between threads is accomplished via shared memory, avoiding specific message passing at the intra-node level. Inter-node communications are carried out by means of messages when one multithreaded process requires data from other nodes. This approach requires introducing additional programming effort to turn a pure shared or distributed memory code into a hybrid code. Such programs are implemented on the basis of a programming model known as mixed mode parallel programming.

The most widespread model for mixed mode programming on multicore clusters lies on the combination of the industry standards OpenMP (for shared memory) and MPI (for distributed memory), so that each MPI process comprises multiple OpenMP threads. At the intra-node level, the processes use OpenMP threads to take advantage of shared memory capabilities within a node, while MPI is

Algorithm 3 Vector addition with MPI+OpenMP

```
1:  #include <mpi.h>
2:  #include <omp.h>
3:  #include <stdio.h>
4:  #define MASTER_ID 0
5:  #define TAG 1
6:  #define MAX_THREADS 8
7:  void vector_addition (int* A, int* B, int* C, int length)
8:  {
9:      int offset, position, i;
10:     int myrank, size;
11:     MPI_Status status;
12:     /* Initialization of the MPI processes */
13:     MPI_Init (NULL, NULL);
14:     MPI_Comm_rank (MPI_COMM_WORLD, &myrank);
15:     MPI_Comm_size (MPI_COMM_WORLD, &size);
16:     /* Message-passing: sending tasks */
17:     if(myrank==0){
18:         initializeInputArrays (A, B, length);
19:         /* Sending blocks of length/size elements to each MPI process */
20:         for(i=1; i<size; i++){
21:             offset = i*(length/size);
22:             MPI_Send(&A[offset], length/size, MPI_INT, i, TAG, MPI_COMM_WORLD);
23:             MPI_Send(&B[offset], length/size, MPI_INT, i, TAG, MPI_COMM_WORLD);
24:         }
25:     }
26:     else{
27:         offset = myrank*(length/size);
28:         MPI_Recv(&A[offset], length/size, MPI_INT, MASTER_ID, TAG, MPI_COMM_WORLD, &status);
29:         MPI_Recv(&B[offset], length/size, MPI_INT, MASTER_ID, TAG, MPI_COMM_WORLD, &status);
30:     }
31:     /* Performing vector addition over the assigned block with OpenMP */
32:     position = myrank*(length/size);
33:     #pragma omp parallel for num_threads(MAX_THREADS) shared(A,B,C,position,length,size) private(i) schedule(static)
34:     for(i=position; i<position+length/size; i++)
35:         C[i] = A[i] + B[i];
36:     /* Message passing: results recovery */
37:     if(myrank==0){
38:         for(i=1; i<size; i++){
39:             offset = i*(length/size);
40:             MPI_Recv(&C[offset], length/size, MPI_INT, i, TAG, MPI_COMM_WORLD, &status);
41:         }
42:     }
43:     else
44:         MPI_Send(&C[offset], length/size, MPI_INT, MASTER_ID, TAG, MPI_COMM_WORLD);
45:     /* Finishing MPI */
46:     MPI_Finalize();
47: }
```

used at the inter-node level to communicate distributed processes located at different nodes. Some of reasons that justify the use of this MPI+OpenMP hybrid approach are the following [53]:

- As some applications show different levels of parallelism, the hybrid approach allows the developer to exploit them under a two-layer approach based on MPI on the upper level (inter-process) and OpenMP on the lower level (intra-process).
- When scalability problems arise in an MPI application (i.e. due to data replication, application requirements or system restrictions), OpenMP can be used to solve such problems and introduce an additional amount of parallelism.

- When load balance issues arise at the MPI level, OpenMP capabilities can be exploited to achieve balanced processing of the workload by using a flexible number of threads per process along with scheduling mechanisms, finer granularity, etc.

- The introduction of OpenMP reduces communication overhead issues within a MPI process, using the shared memory to communicate the threads that compose a MPI process instead of applying explicit message passing.

Algorithm 3 shows an MPI+OpenMP hybrid implementation of the parallel vector addition program. As in the original MPI implementation, the structure of this program defines several MPI processes which operate over a specific portion of the vectors, using point-to-point primitives to communicate the assigned blocks of the input arrays and the obtained results. The key idea behind the mixed mode implementation is to introduce OpenMP directives with the aim of adding multithreading features to the MPI processes. In this way, the processing of the assigned workload is accelerated by taking advantage of shared memory capabilities at the intra-process level. In this example, this is accomplished by using the *#pragma omp parallel for* directive, which initializes a parallel region operating over the immediately following for loop (line 33 in Algorithm 3). Examples of successful applications of MPI+OpenMP to scientific domains include linear seismic processing, [17], atmospheric modelling [108], coastal wave analysis [109], dynamic power balancing [84], and computed tomography processing [93].

4.2.6 Heterogeneous Systems

Current hardware platforms show a multifaceted nature. Systems composed of multicore microprocessors are now complemented by using a wide range of computing devices whose parallel capabilities can be exploited to support the acceleration of time-consuming applications. Examples of this kind of accelerators include digital signal processors, reconfigurable hardware, and graphics processing units (GPUs). Heterogeneous computing [68] aims to provide guidelines for the development of parallel applications on this kind of hardware setups, allowing the programmer to select the best architecture to execute a given task and combining the capabilities of different devices to address the processing of highly computationally demanding workload. The different tasks in a parallel application mix different parallelism opportunities (data parallelism, task parallelism, etc.) and, thereby, accurate distributions of work can be assigned to each device in accordance with their capabilities (SIMD processing, MIMD, etc.). As a result, heterogeneous approaches have shown significant performance when accelerating scientific applications [186].

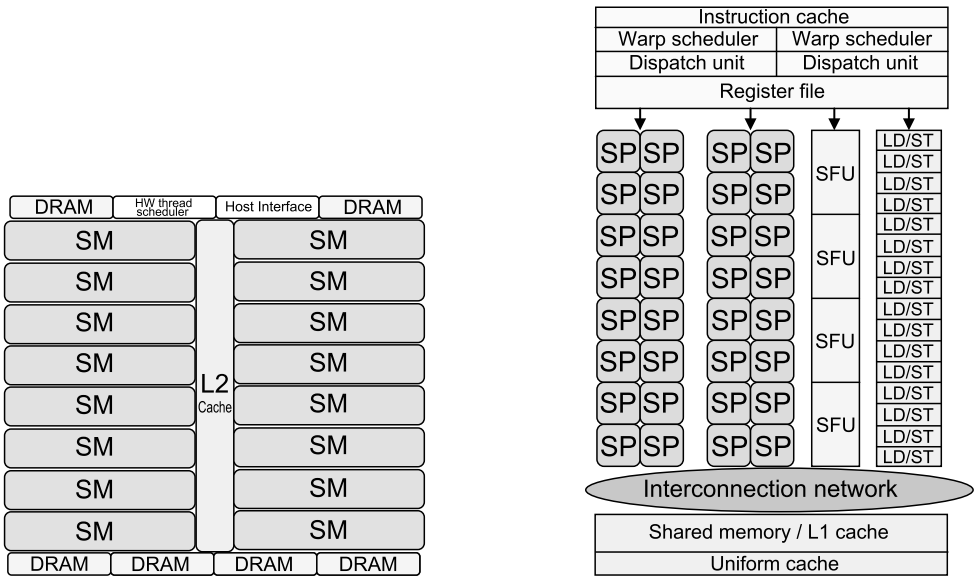
In such heterogeneous context, advances in general purpose computation on GPUs (GPGPU) explain the key role that such computing devices are playing nowadays in the high performance computing area [53]. GPU devices provide new levels of processing capability (mostly focused on highly multithreaded SIMD processing) at a low cost, evolving very quickly into the preferred architecture for dealing with time-consuming applications showing large amounts of data parallelism. By

combining multicore CPUs and GPU-based accelerators, commodity computing platforms provide mechanisms to satisfy the increasing demands of computing power. Moreover, the June 2015 revision of the TOP500 supercomputer list includes more than 50 large-scale systems which are based on this kind of hardware setups [2].

Figure 4.7 represents the Fermi GPU architecture from Nvidia [69]. A Fermi GPU is composed of 16 highly-threaded streaming multiprocessors (SMs). Each SM comprises 32 cores or streaming processors (SPs) that share instruction cache and control logic to perform floating-point and integer operations. Each SP can carry out one floating-point single-precision fused multiply-add operation (FMA) per clock period and one double precision FMA in two periods. Integer ALUs support mathematical and logical operations over 32-bit and 64-bit data values. 16 load-store (LD/ST) units are available within a SM to handle memory operations, along with four special-function units (SFU) to handle special operations, such as *sin*, *cos*, *exp*, and *rcp*. There are four execution blocks per SM, in such a way that a total of 32 instructions can be dispatched to these blocks per cycle. Regarding the memory hierarchy, a local memory of 64K is split into 16K/48K or 48K/16K between cache and shared memory, providing low-latency access to data. An L2 cache covers GPU local DRAM of 768KB for a 512-core configuration, implementing atomic read-modify-write operations to manage the access to shared data. Finally, a graphics double data rate (GDDR) DRAM memory, named as global memory, is available for read/write purposes to the SMs through six 64-bit DRAM channels.

Focusing on a GPU programming perspective, the computational elements of an algorithm executing in a GPU are known as kernels. Kernels consist of multiple threads that execute the same instructions in parallel over different data. These threads are grouped into thread blocks containing up to 1536 threads, while thread blocks are grouped into grids which execute a unique kernel. Each thread block runs in a single SM, allowing the threads inside a block to cooperate through the use of the shared local memory. The term warp refers to each group of 32 threads that compose a thread block, which are managed by the warp scheduler for concurrent execution. The warp represents the fundamental unit of dispatch within a SM, as it executes one common instruction at a time. One of the key ideas to achieve improved performance in a GPU is to make all the threads in a warp follow the same execution path, minimizing branch divergence. An efficient management of the different memory levels is also essential for performance purposes. For example, although the size of the shared memory is smaller than the global memory, we have to consider that global memory accesses are slower than locally shared memory accesses. In addition, memory transactions should be minimized, carefully disposing the data to achieve (if possible) coalesced access patterns and avoiding misaligned or strided accesses within a half warp.

When developing heterogeneous applications, the role of the CPU is not restricted to serve as a host for the accelerator (preparing tasks and obtaining results from the computing device). A multicore multiprocessor can also contribute to the acceleration of the algorithm, performing additional computations or collaborating with the accelerator in the processing of the parallel workload. In fact, modern CPU architectures include in their instruction sets SIMD extensions [81] with the aim of supporting data parallel execution. Examples of these SIMD extensions for x86 processors are



(a) (b)

Fig. 4.7 Fermi GPU architecture, showing a) overview of the device architecture, b) contents of a SM composed of multiple SPs

the different forms of Streaming SIMD Extension (SSE) and Advanced Vector Extensions (AVX). A SIMD vector instruction defines the request of performing a same operation over multiple data elements in parallel. Such instruction allows the processor to perform multiple instances of the same operation with a single instruction, decreasing the amount of scheduling and instruction decode logic. In this way, vectorization strategies complement the multithreaded execution, taking into account that not all the codes are suitable to be vectorized.

4.2.7 Programming Heterogeneous Systems: OpenCL

OpenCL (Open Computing Language) [68] represents the open royalty-free standard for heterogeneous computing, whose specification is managed by the Khronos Group technology consortium (https://www.khronos.org/). OpenCL was proposed to make easy the programming of parallel applications on heterogeneous systems, supporting execution on a wide variety of computing devices, such as multicore CPUs, GPUs, digital signal processors, FPGAs, and other accelerators. This standard defines a framework to develop applications in which a CPU host prepares and assigns tasks to one or more devices that execute kernels to carry out the parallel processing of the workload. For this purpose, OpenCL offers a device-side language specification and a host management layer, providing an API for high-level programming languages such as C and C++. The OpenCL architecture is built upon four interrelated models [68]:

- **Platform model**. It defines an abstract hardware model where one processor (host) interacts with one or more devices composed of one or more processors. In OpenCL terminology, a

computing device is composed of multiple compute units, each one containing a number of processing elements which execute instructions.

- **Execution model**. It specifies how the OpenCL framework is configured at the host side and how kernels are executed on the computing devices. Running an OpenCL kernel code involves concurrent executions of multiple instances of the kernel, which represents the basic executable unit. Kernels are executed by multiple work-items, each one with a unique identifier, which belong to a work-group that executes on a separate compute unit. Synchronizations and communications via shared memory are allowed within a work-group. Kernels are executed in the computing devices upon request from the host side of the OpenCL application, which defines the context where host-device interactions will take place. Given a context with a computing device, commands queues are set up to allow the submissions of orders from the host to the device, including data transfers from the host memory to the device memory and vice versa, kernel execution commands, etc.

- **Memory model**. It defines an abstract memory hierarchy used in kernel execution. The OpenCL memory model defines a global memory which is available for reading and writing purposes by all the work-items in all the defined work-groups, often representing the largest (and slowest) memory in the model. The constant memory defines a region of the memory hierarchy reserved for read-only accesses at the kernel side, remaining its contents constant during the execution of the kernel. A faster shared memory is given by the local memory, which can be used by all the work-items within a work-group. Finally, the private memory is defined at the work-item level, storing the private variables handled by the work-item.

- **Programming model**. The programming model refers to how the concurrency model is mapped to the underlying physical hardware. As OpenCL has been designed to operate with a wide range of computing devices, it supports both data parallelism and task parallelism (although it is especially suitable for SIMD applications).

OpenCL kernels are written by using the OpenCL C programming language. It is based on the ISO C99 specification with some extensions and restrictions. Examples of restrictions include the unavailability of function pointers, bit data types, recursion, variable length array declaration, and support for C99 standard libraries. One of the most important extensions is the availability of support for vector data types and vector operations, which allow the programmer to introduce explicit vectorization strategies into kernels. The portability goal of OpenCL allows any OpenCL code to run on any OpenCL-compatible device. However, performance is not guaranteed to be preserved and the kernels should be implemented according to the characteristics of the underlying hardware.

Algorithms 4 and 5 show OpenCL kernel codes for the vector addition application. Algorithm 4 refers to a kernel designed to execute on a GPU device. The highly multithreaded nature of GPUs leads to an implementation where each work-item processes a single element of the array. To this end, the work-item retrieves its identifier by using the function *get_global_id* (line 4 in Algorithm

Algorithm 4 Vector addition with OpenCL - GPU kernel

```
1: __kernel void vecadd(__global int *A, __global int *B, __global int *C)
2: {
3:    /* Getting the work-item identifier */
4:    int id = get_global_id(0);
5:    /* Adding the corresponding element of A and B */.
6:    C[id] = A[id] + B[id];
7: }
```

Algorithm 5 Vector addition with OpenCL - CPU kernel

```
1: __kernel void vecadd(__global int4 *A, __global int4 *B, __global int4 *C, __const int length)
2: {
3:    /* Getting the work-item identifier */
4:    int id = get_global_id(0);
5:    /* Getting the number of work-items */
6:    int nt = get_global_size(0);
7:    /* Adding the corresponding blocks of int4 elements of A and B */.
8:    for(int i=id*(length/nt); i<(id+1)*(length/nt); i++)
9:       C[i] = A[i] + B[i];
10: }
```

4). The identifier defines the position of the arrays to be added by the work-item (line 6). As can be observed in this example, different reserved words are used to specify a kernel (__kernel) and the location of the arrays in the memory model (in this case, they are stored in the global memory __global). On the other hand, Algorithm 5 shows the implementation of the kernel for a multicore CPU. For the CPU case, each work-item will process a block of elements from the arrays, which have been defined using the vector types int4 to enable explicit vectorization of the code. By using this type, each addition involves four elements of the arrays in a single instruction (line 9 in Algorithm 5). Work-items perform these operations over blocks of int4 elements defined in accordance with their identifiers (line 4) and the overall number of work-items in execution (obtained by using the function *get_global_size*, line 6). More details on OpenCL programming are available in [68].

4.2.8 Parallel Multiobjective Metaheuristics

Multiobjective metaheuristics have been traditionally applied to provide satisfying approximations sets to the Pareto-optimal front in reasonable time, addressing the NP-hard nature of most MOPs. However, some real-world problems are so computationally demanding and the search space so large that even the application of metaheuristics requires high execution times in order to find high-quality solutions. The computation of objective functions often represents the most time-consuming operation in a metaheuristic, showing not only high execution time *per se* but also significant variability in the times required to evaluate two different individuals in accordance with the characteristics of the solutions they encode. These issues explain the increasing interest in developing parallel multiobjec-

tive metaheuristics that take advantage of the computing capabilities of modern parallel architectures. In fact, a whole subfield of research in the multiobjective optimization area deals with the integration of parallel and distributed computing into multiobjective searches. The term pMOEA [184] was introduced to refer to those MOEAs which decompose their computational workload among multiple processors. In spite of being referred to evolutionary designs, it should be noted that this idea can also be applied to other non-evolutionary bioinspired approaches like swarm intelligence. Hence, parallel multiobjective algorithms represent the preferred implementation for solving complex real-world MOPs [37], an statement justified by the significant amount of problems [6, 177] successfully tackled by this kind of developments.

In general terms, parallel computing techniques can be integrated into multiobjective optimizers to decompose the problem and decrease execution time by distributing independent tasks among multiple processors. Parallelism can also be applied to improve the quality of the obtained results, allowing the metaheuristic to explore more regions of the solution space in the same amount of time as the serial counterpart. According to Talbi et al. [177], the parallelization of multiobjective metaheuristics has followed three major hierarchical models:

1. **Self-contained parallel cooperation**.
2. **Problem independent intra-algorithm parallelization**.
3. **Problem dependent intra-algorithm parallelization**.

The main goal in a self-contained parallel cooperation approach is to improve the quality of the generated results in a given execution time. The most representative example of this kind of parallel designs is the island model paradigm. The island model is based on the phenomenon of different (sub)populations evolving separately in relative isolation, with migration taking place to move individuals from one island to another. In these parallel designs, the population of the algorithm is split into several independent and separate (sub)populations known as demes. Each island will be managed by a different processor, which runs the whole multiobjective metaheuristic over its corresponding deme. This parallelization model defines a new evolutionary operator known as migration, which allows the exchange of promising individuals between islands attending to some fitness criteria. Migration policies define how often migration occurs in terms of number of generations, neighbourhoods, the number of individuals involved in the migration procedure, and how to select the individuals which will be removed from the population to allocate the new ones. Under this scheme, gene flow between demes is achieved in a limited way, in order to allow each island to evolve separately. This parallelization scheme is suitable to address MOPs with very large search spaces, partitioning the search space in such a way that many different regions are processed simultaneously. Furthermore, the island model allows different execution scenarios, such as different islands executing the same metaheuristic with different parameters or different islands executing different metaheuristics. Distributed-memory systems are usually chosen to run this kind of parallel metaheuristics, as reduced communications are required (only in the migration procedure, which usually

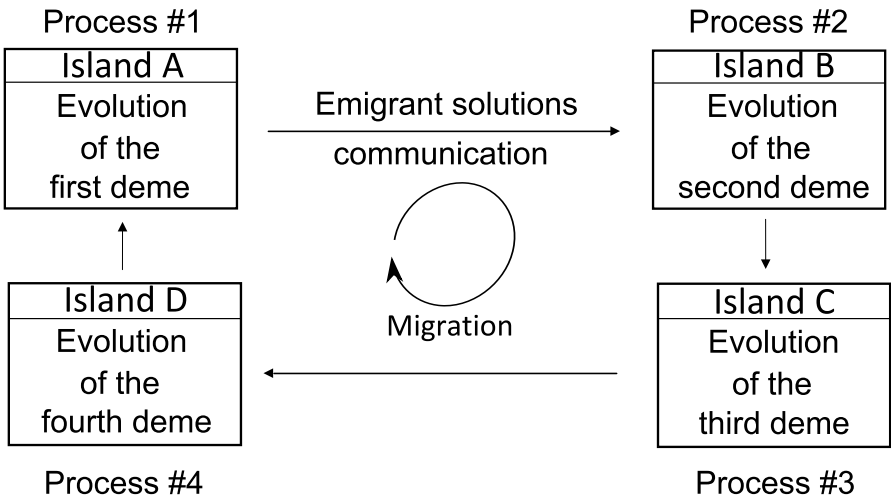


Fig. 4.8 Island model parallelization scheme showing four demes

takes place a limited number of generations). Figure 4.8 shows a graphical example of island model with clockwise migration patterns between islands.

On the other hand, the intra-algorithm parallelization models aim to apply parallelism to reduce the overall execution time of the metaheuristic. The problem independent variant is focused on defining sets of independent tasks that can be distributed among computing units in the architecture throughout the execution of the algorithm. In a population-based multiobjective metaheuristic, the main loops operate over a data structure composed of multiple candidate solutions to the problem. When processing new solutions, the application of evolutionary operators and fitness evaluations can be mapped to different parallel processes (each one processing a different individual), as the same workflow of operations is repeated multiple times without showing dependencies between repetitions. That is, the processing of different new candidate solutions can be performed concurrently. This parallelization scheme leads to reduced execution times while maintaining the search capabilities of the original serial implementation. The most representative example of this intra-algorithm parallelization is the master-worker scheme, where a master process executes population management operations and other non-parallelizable tasks, defining afterwards a number of parallel tasks (i.e. evaluations of new individuals) which are distributed among other parallel processes known as workers. These workers carry out the tasks communicated by the master process (which can also be doing worker tasks or other operations) and return the obtained results (i.e. objective function scores). Such parallelization scheme is graphically represented in Figure 4.9. It is noted that the "problem independent" term refers to the intrinsic parallel opportunities shown by metaheuristics, as a successful parallelization must not ignore in any case the characteristics of the problem, the time profile of the application, and the underlying hardware features (shared/distributed/hybrid shared-distributed memory organization, interconnection networks, number of processing units, etc.).

The problem dependent intra-algorithm variant mostly refers to the explicit parallelization of the objective function loops. In real-world MOPs, evaluation procedures often involve very time-

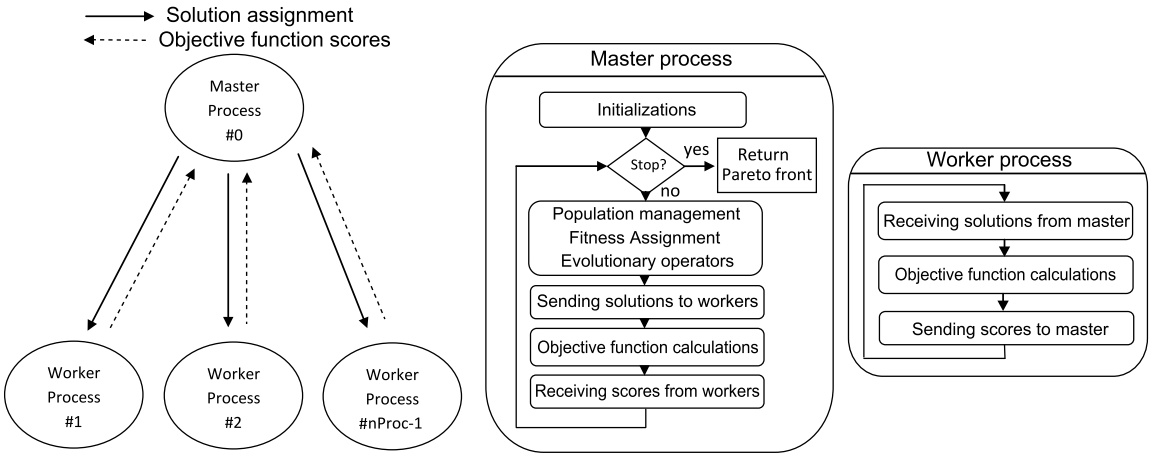


Fig. 4.9 Master-worker intra-algorithm parallelization scheme

consuming operations which show data parallelism opportunities in their processing. This is a very common case in the field of bioinformatics, where a wide range of biological objective functions which show data parallelism at the sequence character level can be found (i.e. the phylogenetic parsimony and likelihood functions). In this way, the objective function computations can be decomposed, giving as a result a high number of operations to be performed in parallel. If the objective function shows such data parallelism opportunities, this parallelization model seems suitable to benefit from the use of accelerators, SIMD extensions, and heterogeneous GPU+CPU computing. In this case, the parallelization must be undertaken according to the characteristics of the objective function, taking into account possible sources of load imbalance and defining proper workload assignments for the computing devices involved in the computations.

In this Thesis, we are interested in analyzing intra-algorithm parallelization schemes for our metaheuristics, with the aim of dealing with the high execution times shown by phylogenetic analyses on real scenarios. Such schemes are implemented by using parallel computing techniques under different programming models (OpenMP, MPI, hybrid MPI+OpenMP) and different hardware architectures (pure shared-memory environments with a high number of processing cores and hybrid shared-distributed memory setups). In addition, the implications of applying heterogeneous computing to parallelize objective functions are explored through the application of OpenCL-based kernel implementations of the phylogenetic parsimony function, studying a CPU+GPU design that combines the capabilities of current heterogeneous systems to address the problem.

4.3 Performance Metrics Used in This Thesis

This section is focused on detailing the different performance metrics used to assess the algorithms studied in this Thesis. Firstly, the multiobjective quality indicators used to examine the quality of the obtained Pareto approximation sets are introduced. Afterwards, parallel metrics for measuring the

relevance of the analyzed parallelization strategies are summarized. Finally, biological metrics for model selection and phylogenetic tree quality measurement are described.

4.3.1 Multiobjective Performance Assessment

As pointed out in Section 4.1.2, the evaluation of multiobjective performance (how good are the approximation sets generated by each metaheuristic) must be performed by using a variety of quality indicators which provide complementary information about solution quality. In this Thesis, the proposed multiobjective metaheuristics are assessed by using three quality indicators: hypervolume [14], coverage relation [194], and spacing [183].

- **Hypervolume**. Hypervolume is a unary quality indicator which defines the quality of the approximation set generated by a multiobjective approach by measuring the region of the objective space Z which it covers. In this way, this metric provides a numerical measurement of the convergence and diversity observed in the generated solutions. Let X be the outcome of a multiobjective algorithm, defined as a set of points in the objective space Z=$\Re^n$. Formally, the hypervolume $I_H(X)$ is defined as the n-dimensional volume of the objective space (with regard to a reference point Z_{ref}) which is weakly-dominated by at least one point $s \in X$, this is, the volume of the hole-free orthogonal polytope $\prod^n$ [14]:

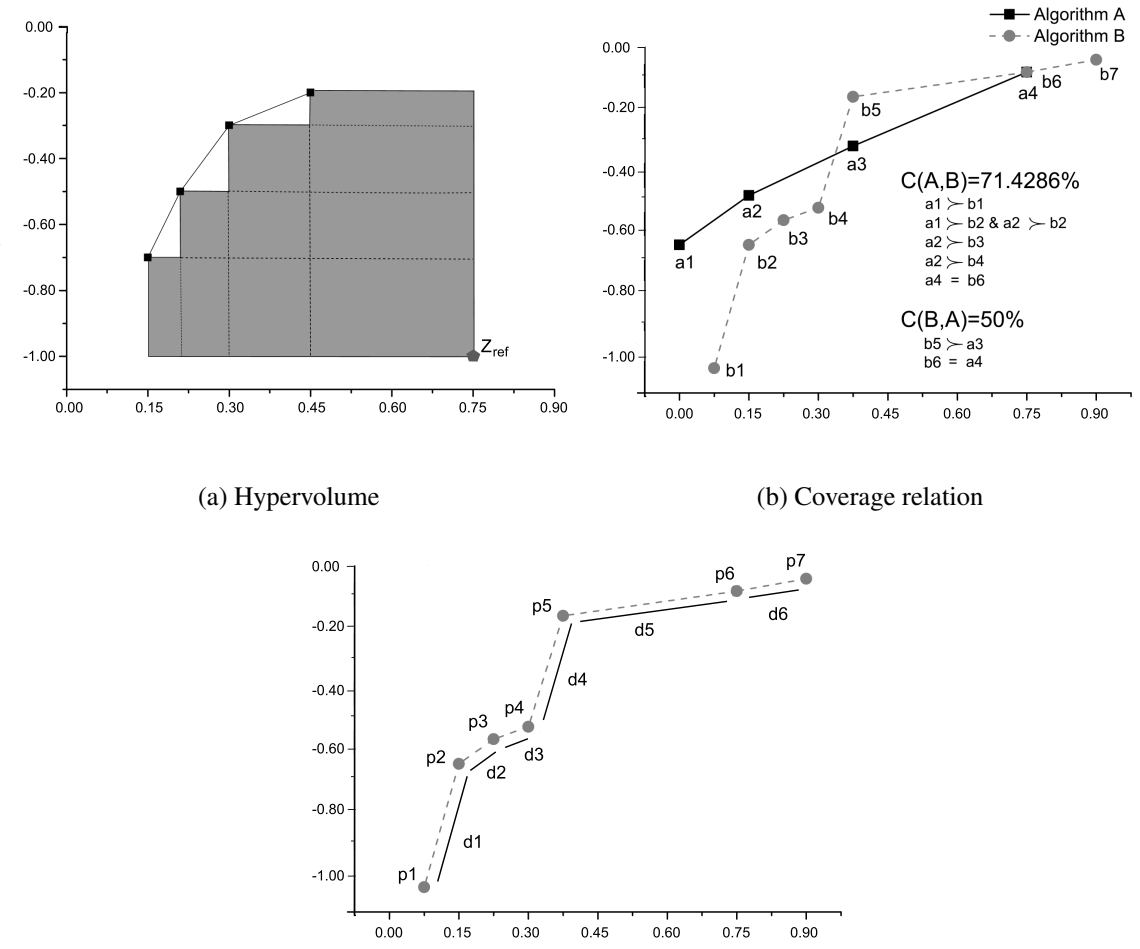
$$\prod^n = \{p \in \Re^n : p \preceq s \text{ for some } s \in X\}. \tag{4.1}$$

  A graphical example of hypervolume is shown in Figure 4.10 (a). For a MOP involving two objectives (as in the problem addressed in this Thesis), the hypervolume indicator measures the area of the objective space covered by the generated solutions. Those approximation sets which *maximize* hypervolume will be preferred, as higher hypervolume values imply a better diversity and convergence to the Pareto-optimal front. A binary extension of this metric was proposed in [195], the I_{HD} indicator, which compares two sets of solutions X and Y as follows:

$$I_{HD}(X,Y) = \begin{cases} I_H(Y) - I_H(X) & \text{if } \forall s \in Y, \exists s' \in X : \\ & s' \succ s, \\ I_H(X+Y) - I_H(X) & \text{otherwise.} \end{cases} \tag{4.2}$$

  In the above expression, $I_H(X)$ and $I_H(Y)$ define the hypervolume of the objective space dominated by X and Y, respectively, so that $I_{HD}(X,Y)$ represents the region dominated by Y but not by X. This binary extension provides a useful tool to introduce hypervolume-based fitness assignment procedures for indicator-based multiobjective metaheuristics. In this work, I_{HD} will be used as the quality metric considered in our implementations of IBEA and IMOBA.

  When assessing multiobjective metaheuristics under this metric, normalized objective scores are usually used to avoid the influence of different objective scales in hypervolume. Hence,



(a) Hypervolume

(b) Coverage relation

(c) Spacing

Fig. 4.10 Quality indicators used to assess multiobjective bioinspired proposals

the scores of the solutions contained in each approximation set will be normalized in the scale [0,1], using for this purpose ideal and nadir points which are defined attending to the scale of the objective functions in the analyzed dataset. Therefore, this indicator will measure the area covered by each normalized front with regard to the reference point $Z_{ref} = (1,1)$, taking both objectives to be minimized by considering positive likelihood values.

- **Coverage relation**. Also known as set coverage, this metric is a binary quality indicator defined to make pairwise comparisons of the outcomes reported by two multiobjective metaheuristics according to Pareto dominance. Given two Pareto approximation sets X and Y, the coverage relation measures the fraction of solutions in Y which are weakly-dominated by X:

$$SC(X,Y) = \frac{|\{y \in Y, \exists x \in X : x \succeq y\}|}{|Y|}. \tag{4.3}$$

Those approaches that maximize *SC* must be preferred, as a *SC*(X,Y)=100% implies that all the points in Y are covered by X. Thereby, this quality indicator allows the decision on which metaheuristic is better from a Pareto dominance perspective attending to the pairwise comparison of the generated solutions. As the set coverage is not a symmetric measurement, both *SC*(X,Y) and *SC*(Y,X) must be computed. Figure 4.10 (b) shows an example of two Pareto fronts compared under the coverage relation indicator for a bi-dimensional MOP.

- **Spacing**. The uniformity of the distribution observed in the obtained Pareto fronts will be evaluated by using the spacing metrics, a diversity-based quality indicator that takes into account the distances between neighbouring points in X:

$$SP(X) = \sqrt{\frac{1}{|X|}\sum_{i=1}^{|X|}(d_i - \bar{d})^2}, \tag{4.4}$$

where d_i is the least Euclidean distance between a point i and any j (in the objective function space), taken as $\sum_{k=1}^{n} |f_k(i) - f_k(j)|$ where f_k is the normalized score at the k-th objective, and $\bar{d}$ is the mean value of all d_i. Lower spacing values suggest a better spread of solutions in the front. Figure 4.10 (c) shows a graphical representation of spacing for a bi-dimensional MOP.

4.3.2 Parallel Performance Assessment

When parallelizing multiobjective bioinspired metaheuristics, it is interesting to assess how the parallelization strategies take advantage of the computing capabilities of the underlying architecture. In order to examine the parallel performance of the proposals, two widely-used parallel metrics are considered [129]: **speedup** and **efficiency**.

Speedup measures the improvement in execution time achieved by the parallel version of an algorithm with regard to the serial version. Let T_1 be the execution time on a single processor reported by an application (serial execution time), and T_n the execution time on n processors. The speedup obtained when running the parallel application on n processors is defined as:

$$Speedup(n) = \frac{T_1}{T_n} \tag{4.5}$$

On the other hand, efficiency provides a measurement of the average utilization of the processing units achieved by a parallel algorithm. That is, how much the parallel algorithm takes advantage of the available hardware resources to reduce execution time. Let *Speedup(n)* be the speedup attained by a parallel application running on n processors, then the achieved efficiency is calculated as:

$$Efficiency(n) = \frac{Speedup(n)}{n} \tag{4.6}$$

Efficiency values can also be computed as a percentage:

$$\%Efficiency(n) = 100 * \frac{Speedup(n)}{n} \tag{4.7}$$

The idea behind using these two metrics lies on obtaining measurements of how the parallel algorithms are able to exploit the different parallel architectures considered in this Thesis. The scalability concept assesses the application's ability to decrease execution time when using a growing number of processors. According to this definition, parallel scalability studies are conducted to verify the behaviour of the examined parallel algorithms on increasing problem sizes (increasing complexity of the input dataset) and system sizes (increasing number of processors/cores). In order to understand how the algorithms benefit from the use of increasing number of processing units, the results reported by these metrics will be discussed by analyzing the different factors which impact parallel performance, such as thread management and waiting times due to load imbalance in shared-memory systems, overhead due to communications and synchronizations in distributed-memory environments, host-device data transfer times in heterogeneous CPU+GPU contexts, etc.

4.3.3 Biological Performance Assessment

Regarding biological performance assessment, we are interested in two main lines of biological performance metrics: evolutionary model selection tests and biostatistical measurements for phylogenetic topology comparison.

Model Selection

A wide range of models of nucleotide evolution have been described in the literature. The likelihood of a phylogenetic hypothesis strongly depends on the assumptions about the evolutionary process represented by the evolutionary model. In order to choose the most accurate model of nucleotide substitution for our analyses, model selection testing procedures are conducted by using the widely-used tool jModelTest [48]. This software analyzes the suitability of the evolutionary model to a given biological dataset by using different statistical approaches. In this Thesis, statistical model selection is carried out by applying two main tests [16]: the **Akaike Information Criterion (AIC)** and the **Bayesian Information Criterion (BIC)**. On the one hand, the AIC test computes the amount of information lost when a specific model is used to approximate the reality of the evolutionary process. More specifically, AIC implements the selection of the best-fit model by comparing the likelihood of a phylogenetic tree under the different studied models, imposing a penalty factor based on the number of parameters included in each model. Such parameters include base frequencies, substitution rates, proportion of invariable sites, rate variation among sites, etc. Let K_Φ be the number of parameters for the model Φ, M the length of the sequences in the input dataset, and $L(\tau, \Phi)$ the likelihood of the hypothesis modelled by a phylogenetic tree τ and Φ. The model to be preferred is the one that

minimizes the AIC value:

$$AIC = -2L(\tau, \Phi) + 2K_{\Phi}. \tag{4.8}$$

On the other hand, the BIC test evaluates models according to Bayesian estimations. BIC provides an approximation to the natural log of the Bayes factor (a measurement for comparing the evidence for two competing models), so that a smaller BIC value implies a better fit of the model to the observed data. From a Bayesian perspective, the model which shows the smallest BIC represents the model with the maximum posterior probability. Given a model Φ, the BIC value is computed as:

$$BIC = -2L(\tau, \Phi) + K_{\Phi} logM. \tag{4.9}$$

These model selection procedures will be used to test the accuracy of three of the most widely-used models of nucleotide evolution: $HKY85+\Gamma$ (Hasegawa-Kishino-Yano) [78], $TN93+\Gamma$ (Tamura-Nei) [178] and $GTR+\Gamma$ [145]. These models include different assumptions about the evolutionary process. For example, $HKY85+\Gamma$ considers that the purine and pyrimidine substitutions have the same transition-transversion ratio. On the other hand, $TN93+\Gamma$ defines two different transition rates for purines and pyrimidines. Finally, $GTR+\Gamma$ models the evolutionary process by considering six substitution rate parameters. Using jModelTest, the fit of the models to each analyzed dataset will be examined, deciding the most accurate framework (including model parameter configuration) to conduct our experiments. More information on model selection is available in [16].

Metrics for Phylogenetic Comparison

With the aim of conducting reliable comparisons of biological performance with other phylogenetic tools, two biostatistical procedures are applied to analyze the quality of the phylogenetic hypotheses inferred by our bioinspired approaches. Firstly, the **Shimodaira-Hasegawa test (SH)** [158] implemented in the DNAPARS software will be used to look for statistically significant differences in phylogenetic topologies attending to the parsimony criterion. Secondly, likelihood trees will be evaluated by using the **approximately unbiased test (AU)** included in the CONSEL package for statistical phylogeny testing [159], which classifies the competing solutions under study attending to their statistical chances of achieving the best likelihood values.

The SH test is an extension of the Kishino-Hasegawa-Templeton test [100], which makes pairwise tree comparisons by studying the mean and variance of the differences in the number of parsimony steps across sites between competing topologies. Kishino-Hasegawa-Templeton reports that the analyzed trees show statistically significant differences when their means exceed a difference of 1.96 standard deviations. Shimodaira and Hasegawa [158] extended this test to allow the comparison of multiple phylogenetic trees at once. In this proposal, variances and covariances of the sums of parsimony steps across sites are calculated for all the analyzed trees. In order to test if the differences between each tree and the one showing the best parsimony score are statistically significant, the parsimony steps for each tree are sampled using the computed covariances and equal means. From

these values, a P-value is calculated according to the fraction of times the difference between the tree values and the lowest number of parsimony steps exceeds that actually observed.

On the other hand, the AU test compares tree quality in terms of AU values, which are computed by using a multiscale bootstrap approach [159]. For each competing phylogenetic tree, K sets of bootstrap replicates of log-likelihood scores (generated by applying statistical techniques of random sampling with replacement) are obtained according to different sample sizes (sequence lengths). By examining this set of replicates, the test calculates the number of times that each phylogenetic tree is selected as the best likelihood solution. From these counts, denoted as C_1, C_2, ..., C_K, bootstrap probabilities per set are computed as C_1/B_1, C_2/B_2, ..., C_K/B_K, where B_1, B_2, ..., B_K are the number of replicates generated per bootstrap set. AU values are then calculated by verifying for all $i = 1$ to K the change in C_i/B_i along the change of M_i/M, where M_i is the bootstrap sample size and M the number of sites per sequence in the input alignment. Therefore, a phylogeny showing a higher AU value denotes a higher statistical chance of representing the best likelihood hypothesis for the input data. Further details on phylogenetic testing methods can be found in [105].

4.4 Data Sets, Hardware and Material Description

This section introduces the biological data sets used to assess the applied multiobjective metaheuristics, along with the hardware setups where our experiments were conducted and other materials. Throughout the development of this Thesis, two different sets of real nucleotide instances have been analyzed. Firstly, the assessment of the parallel multiobjective metaheuristics considered in this research has been conducted over representative samples of real phylogenetic instances, most of them widely-used in the literature for benchmark purposes. These data sets are listed below:

1. **rbcL_55**: rbcL plastid gene dataset [107], containing 55 sequences (1314 nucleotides per sequence).

2. **mtDNA_186**: Human mitochondrial DNA dataset [87], containing 186 sequences (16608 nucleotides per sequence)

3. **HIV1_192**: HIV1 nucleotide data [1], containing 192 sequences (817 nucleotides per sequence).

4. **RDPII_218**: Prokaryotic RNA dataset [38], containing 218 sequences (4182 nucleotides per sequence).

5. **S1482_346**: Plant genus Tiquilia dataset [122], containing 346 sequences (897 nucleotides per sequence).

6. **ZILLA_500**: rbcL plastid gene dataset [30], containing 500 sequences (759 nucleotides per sequence).

An additional alignment to be included in this first set is **M2771_27** [124], a well-known Plethodontid salamander mitochondrial DNA dataset containing 27 sequences of 15778 nucleotides, which has been considered as case study for analyzing the biological implications of multiobjective optimization in phylogenetics. Secondly, the study of heterogeneous computing techniques to accelerate the parsimony phylogenetic objective function has been carried out by conducting experiments over data sets with different combinations of number of sequences and sequence lengths. The idea is to count with a variety of phylogenetic data sets with different characteristics to determine which ones are more suitable to be analyzed by using GPU or CPU strategies:

1. **M8631_50**: Enterobacteriaceae DNA dataset [85], containing 50 sequences (42456 nucleotides per sequence).

2. **M21119_55**: Caecilian Amphibian DNA dataset [115], containing 55 sequences (4675 nucleotides per sequence)

3. **M19259_156**: Salmonella enterica DNA dataset [182], containing 156 sequences (119750 nucleotides per sequence).

4. **M10272_169**: Mammalian genomic data [117], containing 169 sequences (24251 nucleotides per sequence).

5. **ZILLA_500**: rbcL plastid gene data [30], containing 500 sequences (759 nucleotides per sequence).

6. **HIV1_1459**: HIV1 nucleotide data [1], containing 1459 sequences (8610 nucleotides per sequence).

Regarding hardware setups, several system configurations have been considered in accordance with the nature of the undertaken research:

1. **Multiobjective and biological studies**. Experiments were carried out on an AMD Opteron Magny-Cours 6174 processor at 2.2 GHz with 12MB L3 cache and 16GB DDR3 RAM.

2. **Study of shared-memory parallel designs**. This study was performed using two hardware configurations: a computing node composed of two AMD Opteron Magny-Cours 6174 processors (12 cores per processor, a total of 24 cores in the system) at 2.2 GHz with 12MB L3 cache and 32GB DDR3 RAM. The second adopted setup comprises four AMD Opteron Magny-Cours 6174 processors (a total of 48 cores) at 2.2 GHz with 12MB L3 cache, and a total available DDR3 RAM of 64GB.

3. **Study of hybrid shared-distributed memory parallel designs**. This study considers a cluster configuration composed of six computing nodes interconnected by using a gigabit Ethernet network. Each node contains two quad-core Intel Xeon CPU E5410 (48 cores in total) at

2.33 GHz with 12MB L2 cache and an available DDR2 RAM of 8GB. A second constellation configuration involves two computing nodes, each one composed of two AMD Opteron Magny-Cours 6174 processors (a total of 48 cores) at 2.2 GHz with 12MB L3 cache and 32GB DDR3 RAM, also interconnected by using gigabit Ethernet.

4. **Study of heterogeneous CPU+GPU systems**. Two different heterogeneous setups were considered in this study. Firstly, a machine comprising an Intel i7 950 CPU at 3.07GHz with 12GB RAM (8 compute units - 4 physical CPU cores with hyperthreading), and two GPUs NVIDIA GTX 580, each one running at a GPU Clock rate of 1.59GHz and 1.54GHz, respectively (Fermi architecture, 16 compute units - streaming multiprocessors and 512 GPU cores). The second setup involves an Intel i7 4770K CPU at 3.5GHz with 32GB RAM (8 compute units - 4 physical CPU cores with hyperthreading), a first GPU NVIDIA GTX 780 Ti running at a GPU Clock rate of 1.05GHz (Kepler architecture, 15 compute units - streaming multiprocessors, 2880 GPU cores), and a second GPU NVIDIA Tesla K40c running at 0.75GHz (Kepler architecture, 15 compute units - streaming multiprocessors, 2880 GPU cores).

As for software configurations, the machines used for the three first types of study include Scientific Linux 6.6 as operating system, the GCC 4.4.7 compiler, and the MPI implementation MPICH 3.1.2. The first heterogeneous setup examined in the last study runs an openSUSE OS 12.1, using GCC 4.6.2 and the Intel OpenCL 1.2 and NVIDIA OpenCL 1.1 implementations for compilation purposes. Finally, the second heterogeneous machine uses openSUSE OS 13.1, GCC 4.8.1, and the Intel OpenCL 1.2 and NVIDIA OpenCL 1.1 implementations.

As introduced in the main goals of the Thesis, the relevance of the obtained results is directly related to the relative performance obtained by the analyzed algorithmic approaches with regard to the state-of-the-art. For biological comparison purposes, we have considered different phylogenetic tools from other authors. In summary, three main classes of tools are considered: multiobjective methods, single-criterion parsimony tools, and single-criterion likelihood tools.

Firstly, PhyloMOEA [24] has been used as representative example of multiobjective methods for phylogenetic reconstruction. PhyloMOEA is a multiobjective proposal published by Cancino and Delbem to conduct multiobjective phylogenetic analyses according to the parsimony and likelihood principles. The algorithmic design of PhyloMOEA is based on NSGA-II, considering a tree-shaped individual representation and evolutionary operators which involve the application of PDG and NNI movements for crossover and mutation purposes.

Secondly, parsimony trees have been compared with two methods, TNT [72] and DNAPARS [58]. TNT is the current reference tool for conducting parsimony analyses. It combines different well-known heuristics (such as sectorial search and tree fusing) to address the inference of maximum parsimony trees. DNAPARS is a classic tool from the PHYLIP package which has been widely-used in the literature to carry out nucleotide-based parsimony searches.

Finally, the evaluation of likelihood results has been performed by making comparisons with five methods: RAxML [171], IQPNNI [120], MrBayes [148], Garli [200], and MetaPIGA [80]. The

main characteristics of these tools are described next. Firstly, RAxML is one of the state-of-the-art methods for carrying out analyses according to the likelihood criterion, combining in its design several techniques such as simulated annealing, rapid hill climbing, and lazy subtree rearrangements. IQPNNI is a likelihood method based on a quartet puzzling approach to reconstruct intermediate trees which are optimized by using the NNI heuristic. A Markov Chain Monte Carlo method represents the main core in MrBayes, a popular tool for Bayesian inference. The two remaining methods, Garli and MetaPIGA, are evolutionary algorithms which aim to adapt the classic genetic algorithm scheme to phylogenetics. For this purpose, Garli uses a tree-based individual representation to perform searches directly over the tree space, applying a mutation operator which considers different topological rearrangement heuristics (NNI and SPR) and branch length optimization. Regarding MetaPIGA, it defines a metapopulation genetic algorithm which operates by managing several tree populations which interchange topological information to guide the inference process.

For parallel comparison purposes, we have considered four main parallel methods which count with multithreaded and mixed mode implementations: PhyloMOEA, RAxML, IQPNNI, and MrBayes. In addition, the acceleration of the phylogenetic parsimony function on heterogeneous systems has been assessed by making comparisons with the state-of-the-art. Nowadays, the fastest open-source implementation of parsimony is included in Parsimonator [5], a proposal which takes advantage of OpenMP and SSE3/AVX SIMD instructions to exploit the data parallelism shown by this objective function. Therefore, the relevance of the results attained in this Thesis will be discussed by considering this wide range of methods from the literature.

4.5 Experimentation and Statistical Methodology

When evaluating the performance of parallel bioinspired algorithms, we must take into account that their execution times can be affected by their stochastic nature and external factors which include the background tasks and daemons managed by the operating system. Moreover, variability in the outcome of the algorithm may be observed in different runs according to the stochastic features that characterize bioinspired search strategies. These two issues motivate the assessment of the examined proposals by conducting independent sets of experiments following statistical procedures. For this reason, our comparative studies of parallel and multiobjective performance will be carried out by considering the results reported from 31 independent runs per dataset of each algorithm. In the case of parallel scalability studies, experiments have been conducted using different problem sizes (different data sets) and system sizes (different number of cores used according to the platform). Once again, 31 independent runs per experiment have been considered to attain representative samples of execution time for each dataset and system configuration. The parallel and multiobjective results reported in this research correspond to the median results observed in each set of experiments.

In order to make statistically-reliable analyses, results samples from each metaheuristic have been studied by using the following statistical methodology [157]. Firstly, we carried out the Kolmogorov-Smirnov normality test to check if the evaluated samples followed a Gaussian distribution. In

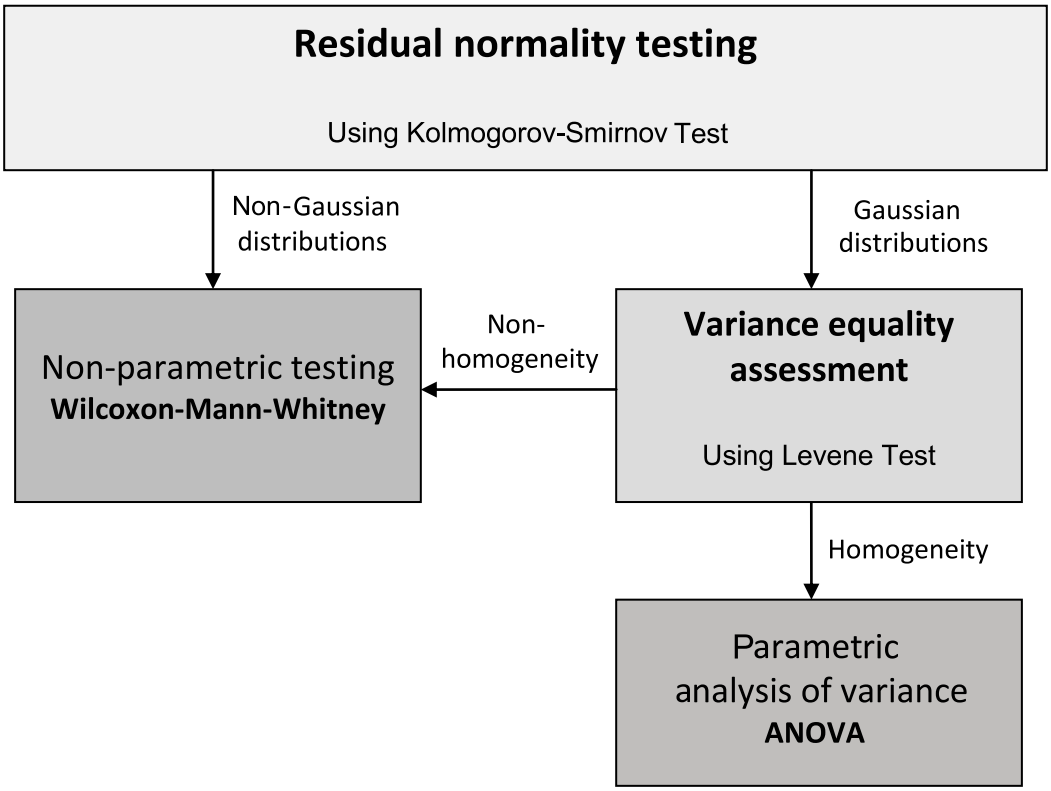


Fig. 4.11 Statistical testing methodology used in this Thesis

accordance with the output of this first step, statistically significant differences in the tested samples were examined in the following way. For non-Gaussian distributions, the Wilcoxon-Mann-Whitney test was applied. On the other hand, if the output of Kolmogorov-Smirnov suggested Gaussian distributions, the Levene test was used to perform homoscedasticity studies. If Levene detected homogeneity in variances, the ANOVA test was conducted. Otherwise, we applied Wilcoxon-Mann-Whitney. In these tests, we applied a confidence level of 95%, with the aim of ensuring that differences were unlikely to have taken place by chance with a probability of 95%. Figure 4.11 shows a graphical representation of the statistical testing methodology followed in this research.

4.6 Summary

This chapter has given account of the methodology used throughout the development of this Thesis, introducing the basis of multiobjective optimization, bioinspired computing, and parallelism. The first main section of this chapter has addressed the explanation of some common notions in multiobjective optimization. The hardness of finding the Pareto-optimal set in real-world MOPs explains the motivation on developing approximated methods based on bioinspired computing. These approaches are built upon search strategies inspired by nature, such as evolutionary computation mechanisms and swarm intelligence. Focusing on the multiobjective metaheuristic design issue, two main groups of

algorithmic designs are distinguished according to the way the fitness assignment procedures are conducted: dominance-based approaches and indicator-based approaches.

In the second main section, we have provided insight into parallel computing paradigms. Starting from a overview of Flynn's taxonomy, details on the different parallel architectures considered in this Thesis have been discussed. Firstly, the main features that characterize shared-memory systems have been introduced, along with the OpenMP multithreading programming model. Secondly, the message-passing organization implemented in distributed-memory architectures has been described, along with an explanation on the MPI programming standard. In this sense, current cluster architectures go a step further by combining the parallel capabilities of both shared and distributed memory paradigms, relying on a mixed mode programming model which allows the development of MPI+OpenMP applications. Nowadays, commodity systems comprise a wide variety of computing devices including multicore CPUs, GPUs, and other accelerators, whose exploitation requires the application of heterogeneous computing techniques. OpenCL provides a framework for the development of parallel applications on this kind of systems, defining different abstraction layers to enable any OpenCL-compatible accelerator to run kernels according to the specification of this standard.

In the final sections, this chapter has addressed the explanation of the multiobjective, parallel, and biological performance metrics used to examine the success of the proposed metaheuristics when tackling the phylogenetic inference problem. Afterwards, the biological data sets analyzed throughout this Thesis have been described, along with the hardware platforms where our experiments were performed and the reference phylogenetic tools which have been considered for comparison purposes. Finally, our experimental and statistical testing methodology has been described, defining a statistically reliable framework to support the soundness of the research.

Chapter 5

Multiobjective Metaheuristics for Phylogenetic Inference

This chapter describes the algorithmic designs of the multiobjective metaheuristics studied in this Thesis to tackle the reconstruction of phylogenetic trees. For each implemented algorithm, details on its general design and its adaptation to the problem are provided. Firstly, some common implementation details are introduced, providing insight into the chosen individual representation, search space processing methodology, initializations, and libraries and tools used to configure and develop the metaheuristics. Once these general features have been examined, the explanation on the implemented metaheuristics is undertaken, distinguishing between dominance-based approaches (MOABC, MOFA, NSGA-II, and SPEA2) and indicator-based methods (IMOBA and IBEA).

5.1 General Features

The first issue that must be addressed when using bioinspired metaheuristics is to choose an accurate solution encoding for the addressed problem. When tackling a real-world MOP, the individual representation represents one of the key design decisions, as poor solution representations might lead to a poor processing of the search space. Due to the meaningful results achieved when combined with evolutionary algorithms [137], a distance-based representation of phylogenetic trees has been considered in this research. The key idea behind this encoding was introduced in *Chapter 2 - Section 2.2.4 Exploring the Search Space*. In summary, this representation considers that a solution is encoded by using an *NxN* symmetric matrix m of evolutionary distances, where N is the number of species in the input alignment. A genetic or evolutionary distance $m[i, j]$ describes a floating-point measure of the evolutionary divergence between two species i and j. Graphically, a parallelism between a distance matrix and a phylogenetic topology is illustrated in Figure 5.1, where $m[i, j]$ represents the length of the path which connects the organisms i and j in the phylogeny.

By using this representation, the search engines implemented in our metaheuristics operate over a matrix decision space, applying distance-based evolutionary operators over these matrices in each

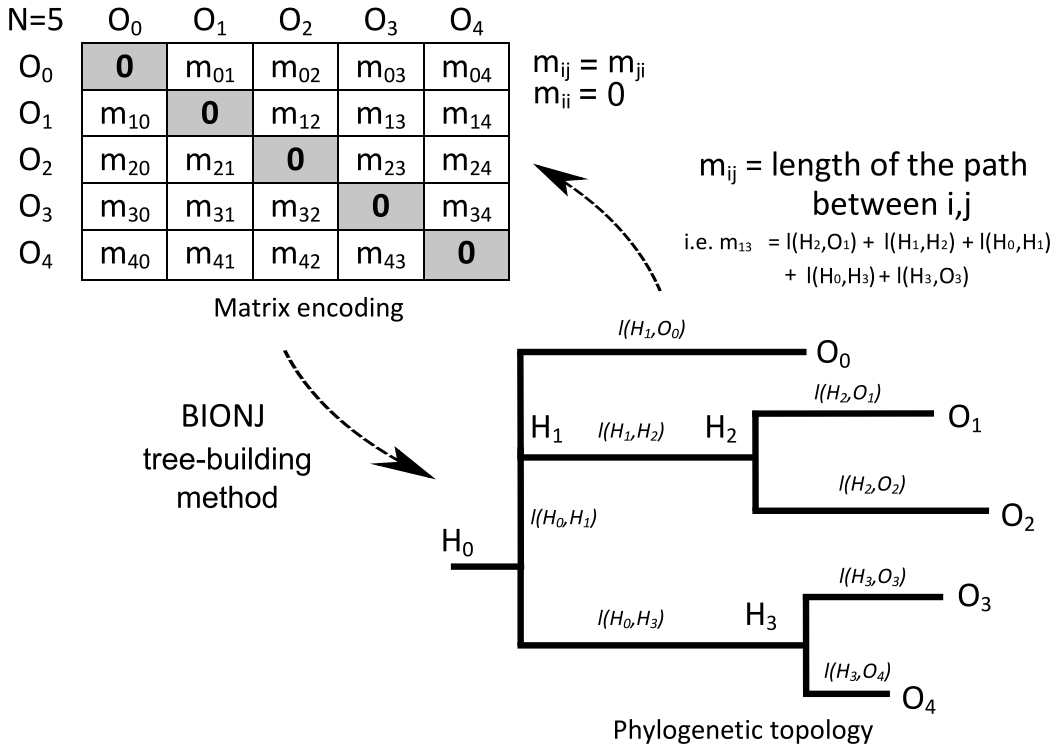


Fig. 5.1 Individual representation: correspondence between a distance matrix and a phylogenetic tree

step of the algorithm. As this representation implements an indirect encoding of solutions, the quality of a candidate solution cannot be assessed from its distance matrix, due to the fact that parsimony and likelihood require as input a phylogenetic topology (as suggested in Equations 2.6 and 2.9 from Chapter 2). Therefore, a mechanism to map the solution from the matrix space to the real phylogenetic tree space is required. For this purpose, the studied metaheuristics include in their implementations the application of the widely-used BIONJ algorithm, originally proposed by Gascuel [67].

As remarked in Chapter 2, BIONJ extends the original NJ concept by using a sampling model of variances and covariances of distances. The idea is to group the closest species i, j attending to the entries of the matrix $m[i, j]$ and some previously computed divergence values $u[i], u[j]$, generating a new partial topology rooted by a node h. Distances are updated in the matrix by replacing the rows corresponding to i and j with a new row which contains the distances from h to the remaining species, using for this purpose a λ factor computed from the observed variances and covariances. These steps are repeated until all the rows in the distance matrix have been processed, generating a full phylogenetic tree. In this way, BIONJ allows the recovering of a phylogenetic tree from its corresponding distance matrix with complexity $O(N^3)$. Furthermore, this method ensures that, for a given phylogenetic topology, there is at least one distance matrix that would make BIONJ reproduce that tree. Algorithm 6 shows BIONJ pseudocode, which is fully detailed in [67].

With the aim of providing improved search capabilities to our metaheuristics, we have taken advantage of the topological search operators defined in the literature. More specifically, a topological

Algorithm 6 BIONJ Algorithm

repeat
  /* *For each row i in m compute the divergence value u[i]* */
  for i = 1 to N **do**
    $u[i] \leftarrow \sum_{j(j\neq i)}^{N} m[i,j]/(N-2)$
  end for
  $i, j \leftarrow$ Select the indices i, j which minimize $m[i,j] - u[i] - u[j]$
  $h \leftarrow$ Create a new parent node and join i, j to it
  /* *Assign branch lengths* bl_i *and* bl_j *from i and j to h* */
  $h.bl_i \leftarrow \frac{1}{2}m[i,j] + \frac{1}{2}(u[i] - u[j])$
  $h.bl_j \leftarrow \frac{1}{2}m[i,j] + \frac{1}{2}(u[j] - u[i])$
  /* *Compute BIONJ* λ *factor* */
  $\lambda \leftarrow$ Compute λ from variances and covariances
  /* *Compute distances from h to the remaining groups* */
  for k = 1 to N **do**
    $D[k] \leftarrow \lambda m[i,k] + (1-\lambda)m[j,k] - \lambda h.bl_i - (1-\lambda)h.bl_j$ /* $k \neq i, j$ */
  end for
  $m, N \leftarrow$ Update matrix (h, D)
  $\tau \leftarrow$ Add partial phylogeny (h)
until There are no more than two groups to be processed
/* *Connect final nodes u, v by a branch with length* $m[u,v]$ */
$\tau \leftarrow$ Add new branch (u, v, $m[u,v]$)
Return τ

optimization procedure based on the application of the heuristics NNI and SPR is included. This topological search step proceeds as follows. According to the mutation rate parameter defined in the metaheuristics, random nodes from the considered phylogenetic topology *currentTree* are selected. For each one, a parametric distance d is calculated as $d = v_{max} - 1$, where v_{max} is the longest distance between the selected node and the leaf nodes (in terms of generations/tree levels). Afterwards, neighbour topologies *neighbourTrees* are generated according to the progressive neighbourhood principle [70]. Firstly, neighbour topologies are generated by pruning and regrafting the selected subtrees on branches located at d generational steps. Further steps involve the application of SPR moves over branches located at $d-1$ decreasing distances and so on, until NNI moves are considered ($d = 1$). The best tree found throughout the execution of these steps is returned as the result of the optimization method. Over the obtained phylogenetic tree, a conjugate gradient method is applied to optimize its branch lengths. Finally, the distance matrix associated to the resulting tree is updated. Algorithm 7 summarizes this topological optimization procedure.

In conclusion, our distance-based methodology considers that the metaheuristic operates over evolutionary distances, generating new matrices which will be mapped to the tree space and evaluated according to parsimony and likelihood. The main goal is to maintain a set of Pareto solutions which evolve throughout the execution of the algorithm, giving as outcome high-quality phylogenies considering multiple criteria (parsimony and likelihood) simultaneously.

After defining the solution encoding and the distance-based methodology followed by our metaheuristics, we have to describe how the starting point of the search is defined. That is, how initial solutions are generated. Our initialization procedure involves the following tasks. Firstly, starter phylogenetic topologies are randomly selected from a repository containing 1000 phylogenies. This tree repository has been generated by performing bootstrap analysis [105] on the input dataset. More

Algorithm 7 Topological optimization procedure

```
1:  neighbourTrees ← 0
2:  /* Choosing random nodes from the original topology */
3:  selectedNodes ← Choose Random Nodes (currentTree.τ, mutationRate)
4:  for i = 1 to |selectedNodes| do
5:      /* Applying progressive neighbourhood search */
6:      d ← Compute Longest Distance (selectedNodes[i], currentTree.τ.leaves)
7:      for j = d to 1 do
8:          if j > 1 then
9:              neighbourTrees ← neighbourTrees ∪ Generate SPR Neighbour (currentTree.τ, j)
10:         else
11:             neighbourTrees ← neighbourTrees ∪ Generate NNI Neighbour (currentTree.τ, 1)
12:         end if
13:     end for
14: end for
15: /* Retrieving the best topology found throughout the process */
16: for i = 1 to |neighbourTrees| do
17:     improves ← Test Topological Move (currentTree.τ, neighbourTrees[i].τ)
18:     if improves == 'true' then
19:         currentTree.τ ← neighbourTrees[i].τ
20:     end if
21: end for
22: /* Optimizing branch length values and updating distance matrix */
23: currentTree.τ ← Optimize Branch Lengths (currentTree.τ)
24: currentTree.m ← Update Distance Matrix (currentTree.τ)
25: Return currentTree
```

specifically, 1000 bootstrap samples of the alignment were computed by using the seqboot tool [58]. These samples were taken as input to generate the topologies contained in the repository, using the parsimony and likelihood classes from the BIO++ bioinformatics library for C++ [75] to infer 500 topologies under maximum parsimony and 500 under maximum likelihood. Parsimony and likelihood scores are computed for each starter solution to verify its quality. After this tree selection step, the next task in the initialization deals with the generation of distance matrices. From the selected topologies, the corresponding distance matrices are computed by assigning to each entry *m[i,j]* the accumulated value of the branch lengths included in the path that connects the organisms i and j in the topology. After these steps, the definition of the starting point of the search (in terms of initial solutions encoded as distance matrices) is accomplished.

Regarding other implementation details, the applied multiobjective metaheuristics have been developed in C++, using the phylogenetic template classes from BIO++ [75]. In addition, the software jModelTest [48] has been used to configure the evolutionary model for likelihood computations (setting transition-transversion ratios, discrete gamma model parameters for the observed among-site rate variation, nucleotide frequencies, and relative nucleotide substitution rates). Finally, the algorithms implemented in this research consider as stop criterion a maximum number of solution evaluations. This stop criterion is common for all the metaheuristics in order to ensure fair comparisons.

5.2 Dominance-Based Approaches

Pareto dominance-based approaches represent the first class of multiobjective methods considered in this Thesis. As stated in Chapter 4, these designs conduct the search towards good Pareto approx-

imation sets by distinguishing solution quality according to the dominance concept. Given a MOP involving n objectives and two solutions s, s' from the decision space SS, s dominates s' ($s \succ s'$) iff $\forall$ $i \in [1,2,...,n]$, $f_i(s)$ is not worse than $f_i(s')$ and $\exists$ $i \in [1,2,...,n]$ such that $f_i(s)$ is better than $f_i(s')$. Hence, a solution $s^* \in SS$ is said to be Pareto-optimal iff $\neg\exists$ $s \in SS$ such that $s \succ s^*$, that is, s^* is non-dominated with regard to the overall decision space. In this context, the optimization goal is defined in terms of minimizing the distance to the Pareto-optimal front and maximizing the diversity of the generated solutions [196]. To this end, Pareto ranking and density estimation mechanisms are included in the fitness assignment procedures to assess the solutions managed by the algorithm.

As representative designs of dominance-based approaches, four bioinspired multiobjective metaheuristics are considered: two swarm intelligence algorithms (MOABC and MO-FA) and two evolutionary algorithms (NSGA-II and SPEA2).

5.2.1 Multiobjective Artificial Bee Colony Algorithm - MOABC

Honey bees in nature are governed by a number of rules which allow them to share information about food sources to guarantee the survival of the hive [153]. The division of tasks in honey bee swarms is organized by defining three different classes of bees. Those bees which exploit the food sources currently known by the hive are called employed bees. When exploiting these sources, employed bees check the environment for promising food sources in the neighbourhood, retaining useful information which is transmitted to onlooker bees in the dance area of the hive. By means of waggle dances, employed bees give account of the quality of the food sources available in the neighbourhood. After observing waggle dances, onlooker bees choose and exploit the most profitable food sources. On the other hand, exploration tasks are conducted by scout bees, which search for new undiscovered food sources in the environment when exhausted sources are detected.

The ABC algorithm, proposed by Karaboga and Basturk [95], models this natural organization to address numerical optimization problems, identifying food sources as possible solutions and nectar amounts as fitness values. A multiobjective adaptation of this algorithm, MOABC, is proposed in this research to conduct phylogenetic searches under parsimony and likelihood. MOABC is a population-based algorithm which manages a swarm (population) P of bees, each one encoding a candidate solution to the problem. The input parameters of this algorithm include the maximum number of evaluations set as stop criterion (*maxEvaluations*), the size of the population (*popSize*), the mutation rate to be considered when generating new solutions (*mutationRate*), and a limit control parameter (*limit*) which defines the maximum number of iterations a solution is allowed to remain in the population (without being improved) until it becomes exhausted. Algorithm 8 shows the pseudocode of the MOABC algorithm.

Initially, bees in the swarm are organized by assigning employed roles to the first *popSize*/2 individuals, while the remaining *popSize*/2 ones will be onlooker bees. Starter solutions (distance matrices) for the employed bees are assigned by using the initialization procedure explained in Section 5.1. (line 2 in Algorithm 8). The main loop in MOABC (lines 3-31) encloses the tasks to

Algorithm 8 MOABC Pseudocode

Input: *maxEvaluations* (maximum number of evaluations), *popSize* (number of bees in the swarm), *mutationRate* (mutation probability used to generate new solutions), *limit* (maximum number of trials a solution can remain until it is replaced by a scout bee).
Output: *ParetoFront* (non-dominated solutions found by the algorithm).

```
1:  ParetoFront ← 0
2:  P ← Initialize Population (popSize/2, dataset)
3:  while ! stop criterion reached (maxEvaluations) do
4:      /* Employed bees exploitation step */
5:      for i = 1 to popSize/2 do
6:          P'_i.m ← Generate Neighbour Employed Matrix (P_i.m, P, mutationRate)
7:          P'_i.τ ← Infer Phylogenetic Tree (P'_i.m, dataset)
8:          P'_i.scores ← Evaluate Solution (P'_i.τ, dataset)
9:          P_i ← Update Employed Solution (P_i, P'_i)
10:     end for
11:     /* Modelling interactions between employed and onlooker bees*/
12:     P ← Apply Fast Non Dominated Sort and Crowding Computation (P, popSize/2)
13:     probabilityVector ← Compute Selection Probabilities (P, popSize/2)
14:     /* Onlooker bees exploitation step */
15:     for i = (popSize/2)+1 to popSize do
16:         P_i ← Select Employed Bee (probabilityVector, P)
17:         P'_i.m ← Generate Neighbour Onlooker Matrix (P_i.m, P, mutationRate)
18:         P'_i.τ ← Infer Phylogenetic Tree (P'_i.m, dataset)
19:         P'_i.scores ← Evaluate Solution (P'_i.τ, dataset)
20:         P_i ← Update Onlooker Solution (P_i, P'_i)
21:     end for
22:     /* Scout bees exploration step */
23:     for i = 1 to popSize do
24:         if P_i.counter > limit then
25:             P_i ← Generate New Scout Solution (P_i, mutationRate)
26:             P_i.scores ← Evaluate Solution (P_i.τ, dataset)
27:         end if
28:     end for
29:     P ← Sort Solutions for the Next Generation (P)
30:     ParetoFront ← Update Pareto Front (P, ParetoFront)
31: end while
```

be performed by the different classes of bees, which define a cooperative exploration-exploitation engine to address the search for high-quality Pareto approximation sets.

Employed bees represent the first class of bees modelled in the algorithm. These bees perform an exploitation step over the solutions known by the hive. For this purpose, employed bees check the neighbourhood of their currently assigned solutions to generate new candidate matrices (lines 4-10). Let P_i be the solution exploited by an employed bee i with distance matrix $P_i.m$, and $P_j.m$ the matrix for a randomly chosen neighbour bee j. A new candidate matrix $P'_i.m$ is generated as follows [95]:

$$P'_i.m[u,v] = P_i.m[u,v] + \phi(P_i.m[u,v] - P_j.m[u,v]), \tag{5.1}$$

where ϕ refers to a random number from a uniform distribution generated in the interval [-1, 1]. As the resulting distance matrix must be symmetric, those entries u, v which verify that *m[u,v]* $\neq$ *m[v,u]* are modified by applying BLX-α as repair operator [137]. Considering $d = |m[u,v] - m[v,u]|$, BLX-α randomly generates a new distance value within an interval $[x_1, x_2]$, where $x_1 = min(m[u,v], m[v,u]) - \alpha d$ and $x_2 = max(m[u,v], m[v,u]) + \alpha d$ (by experimentation, a α value = 1.0 was found to be the one that reported the best behaviour). Once the new candidate matrix has been generated, its corresponding phylogenetic topology $P'_i.\tau$ is computed, along with its parsimony and likelihood scores. After

the inference and evaluation of the new solution, the employed bee must decide which one should be retained (line 9). By using a greedy approach, the employed bee i will give preference to P_i' if it dominates P_i, storing this new solution in its memory. Otherwise, P_i will remain, increasing its associated trial counter, which measures the number of iterations the solution has not been improved.

Next, MOABC undertakes the probabilistic selection and exploitation of the best solutions found in the previous step, modelling the behaviour of onlooker bees. Firstly, the algorithm defines interactions in the dance area between employed and onlooker bees (lines 11-13). To this end, a selection probability vector is computed after sorting the employed bee solutions by means of two operators taken from NSGA-II [52]: fast non-dominated sort and crowding distance ordering. The fast non-dominated sort classifies solutions in several Pareto fronts F_0, F_1, F_2, ... where F_0 represents the non-dominated front, F_1 the front dominated by the solutions in F_0, F_2 the front dominated by F_0 and F_1, and so on. These fronts are ordered internally by estimating the density of solutions surrounding each point in the front. Crowding distance measures this density, giving preference to less crowded solutions. Once employed solutions have been ordered, the selection vector is computed by assigning probabilities to each solution according to its position in the sorted array.

Onlooker bees use this vector to carry out an additional exploitation step over the best solutions found according to the selection probabilities (lines 14-21). By verifying these probabilities, onlooker bees choose the most satisfying solutions, generating new candidate matrices by mutating randomly chosen entries from the selected matrices as follows. Let P_{sel} be the solution selected by an onlooker bee with distance matrix $P_{sel}.m$. A new distance matrix $P_{new}.m$ is mutated from P_{sel} by computing the following expression:

$$P_{new}.m[u,v] = P_{sel}.m[u,v] + \phi(\alpha P_{sel}.m[u,v]), \tag{5.2}$$

where α refers to a gamma-distributed factor based on the genetic rate variation observed in the input dataset [107]. This expression is applied over a limited number of entries u, v, which are randomly chosen with a *mutationRate* probability. The resulting matrices are symmetrized (BLX-α) and their corresponding phylogenetic trees inferred. To promote solution diversity, an onlooker bee will save in its memory the new generated solution if it is not dominated by the originally selected one.

After performing the employed and onlooker steps, the algorithm proceeds with the scout bee exploration step (lines 22-28). In nature, when food sources become exhausted, scout bees are sent to check the environment for new food sources to avoid the starvation of the hive. In MOABC, an exhausted solution is identified by checking if its trial counter has surpassed the value given by the *limit* control parameter. Such solutions are discarded and replaced by the ones generated by scout bees. For each abandoned solution, a new starter topology is randomly selected from the initial tree repository, performing topological optimization in order to allow the new solutions to compete with the ones previously found by the swarm. To this end, we apply the optimization procedure described in Section 5.1. These steps model the exploration step in MOABC, giving as a result new solutions which will be exploited by employed and onlooker bees in the following generations.

After these three main steps, the population is sorted (fast non-dominated sort + crowding distance) in order to assign the *popSize*/2 most promising topologies to employed bees for their exploitation in the next generation. Finally, the *ParetoFront* structure is updated with the non-dominated solutions found in the current generation, starting a new one. After the stop criterion is satisfied, the algorithm returns as output the solutions contained in *ParetoFront*.

5.2.2 Multiobjective Firefly Algorithm - MO-FA

Fireflies in nature are governed by a communication system based on their bioluminescence capabilities, which allow them to attract potential partners for mating purposes. Interactions among fireflies are built upon the concept of light emission, in such a way that those fireflies with the most attractive flashing light patterns will be preferred by other fireflies in the swarm. More specifically, fireflies move towards the position of those fireflies which emit the brightest flashing lights, whose attractiveness depends on several factors, such as the distance between fireflies, the light intensity, and the degree of light absorption by the environment.

Algorithm 9 MO-FA Pseudocode

Input: *maxEvaluations (maximum number of evaluations)*, *popSize* (number of fireflies in the swarm), β_0 (attractiveness factor), γ (environmental absorption coefficient), α (randomization factor).
Output: *ParetoFront* (non-dominated solutions found by the algorithm).

```
1:  ParetoFront ← 0
2:  P ← Initialize Population (popSize, dataset)
3:  while ! stop criterion reached (maxEvaluations) do
4:      /* Firefly movement loop */
5:      for i = 1 to popSize do
6:          for j = 1 to popSize do
7:              if P_j ≻ P_i then
8:                  /* Applying attraction step on P_i */
9:                  δ_ij ← ||P_i.m − P_j.m||
10:                 P_i.m ← Move Firefly (P_i.m,P_j.m,δ_ij,β_0,γ,α)
11:             end if
12:         end for
13:         if P_i has been updated then
14:             P_i.τ ← Infer Phylogenetic Tree (P_i.m, dataset)
15:             P_i.scores ← Evaluate Solution (P_i.τ, dataset)
16:         end if
17:     end for
18:     ParetoFront ← Update Pareto Front (P, ParetoFront)
19: end while
```

This natural behaviour can be useful to design new algorithmic approaches for tackling optimization problems, as suggested by Yang in his swarm intelligence proposal, FA [190]. This population-based metaheuristic identifies the light intensity of a firefly with the quality of its related solution. Brighter fireflies will be associated to better solutions to the problem, making the firefly swarm move towards high-quality solutions by establishing interaction patterns which model this bioluminescence system. Here, we propose MO-FA, a multiobjective adaptation of this novel algorithm. MO-FA extends FA design to resolve MOPs, introducing the dominance concept to decide which fireflies show the most attractive light patterns, that is, the best solutions from a multiobjective perspective. Along with the number of fireflies in the population (*popSize*), MO-FA is ruled by three own input param-

eters which are defined in accordance with the factors that determine the behaviour of fireflies: an attractiveness factor (β_0), an environmental light absorption coefficient (γ), and a randomization factor (α) which introduces variability into firefly movements. Algorithm 9 shows MO-FA pseudocode.

The algorithmic scheme of MO-FA proceeds as follows. First of all, the usual initialization mechanisms are applied to obtain the starter distance matrices associated to the fireflies in the swarm. Once the initial solutions have been generated and evaluated, the main loop in MO-FA (lines 3-19 in Algorithm 9) simulates the attraction system in a firefly swarm P as follows. At each generation of the algorithm, fireflies are compared with each other, in order to detect which fireflies are associated to dominated solutions and which fireflies represent the brightest (non-dominated) ones. Each dominated firefly is then updated with the knowledge provided by the most promising phylogenies generated by the algorithm (lines 6-12). Let P_i and P_j be two fireflies with distance matrices $P_i.m$ and $P_j.m$. If $P_j \succ P_i$ (line 7), P_i will move towards the position (solution) of P_j by computing, in a first step, the overall distance δ_{ij} that separates P_i from P_j (line 9):

$$\delta_{ij} = \sqrt{\sum_{u=1}^{N}\sum_{v=1}^{u}(P_i.m[u,v] - P_j.m[u,v])^2}. \tag{5.3}$$

As evolutionary distances may be significantly different from one dataset to another, δ_{ij} is normalized to the interval [0,10] to avoid widely spread overall distance values. Once δ_{ij} has been calculated, $P_i.m$ is updated (line 10) by using a movement formula which takes into account the attractiveness factor β_0, the absorption coefficient γ, and the randomization factor α. Equation 5.4 specifies how this updating step is carried out:

$$P_i.m[u,v] = P_i.m[u,v] + \beta_0 e^{-\gamma\delta_{ij}^2}(P_j.m[u,v] - P_i.m[u,v]) + \alpha(rand[0,1] - \frac{1}{2}), \tag{5.4}$$

where rand[0, 1] is a random number from a uniform distribution generated in the interval [0, 1]. The second term in this expression denotes the degree P_i will move towards P_j according to the distance which separates them and the attractiveness and environmental absorption factors. The third term introduces some randomness to the movement of P_i, which helps to promote the population diversity and the processing of undiscovered regions of the search space. After updating $P_i.m$, the repair operator BLX-α is applied to preserve the symmetry of the resulting matrix. As P_i can be attracted (that is, dominated) by multiple fireflies in the swarm, the movement loop repeats the above steps for all the fireflies in P that dominate P_i.

The updated distance matrix stored in the firefly under movement is used afterwards to infer the corresponding phylogenetic topology via BIONJ, conducting in addition the topological optimization procedure. The generated phylogeny is then evaluated attending to the parsimony and likelihood criteria. These movement-inference steps are repeated until all the dominated fireflies have been processed. The final task in a generation involves the update of the *ParetoFront* structure according to the current non-dominated solutions. After reaching the maximum number of evaluations, *ParetoFront* is returned as output of the algorithm.

5.2.3 Fast Non-Dominated Sorting Genetic Algorithm II - NSGA-II

NSGA-II is one of the most widely-used metaheuristics for multiobjective optimization. This population-based MOEA, originally proposed by Deb et al. [52], extends the previous NSGA proposal by using the following mechanisms: a fast non-dominated sorting mechanism that assigns Pareto rank values to solutions according to the dominance concept, a crowding distance procedure for point density estimation purposes, and a crowded-comparison operator to guide the selection stage. It takes as input a number of parameters commonly used in evolutionary computation, such as the number of individuals in the population (*popSize*), the crossover probability (*crossoverProb*), and the mutation probability (*mutationProb*). Algorithm 10 shows the pseudocode for NSGA-II.

Algorithm 10 NSGA-II Pseudocode

Input: *maxEvaluations* (number of evaluations), *popSize* (number of individuals in the population), *crossoverProb* (crossover probability), *mutationProb* (mutation probability).
Output: *ParetoFront* (non-dominated solutions found by the algorithm).

```
1:  C, Q, ParetoFront ← 0
2:  P ← Initialize Population (popSize, dataset)
3:  while ! stop criterion reached (maxEvaluations) do
4:      /* Ranking and crowding ordering */
5:      C ← P ∪ Q
6:      Fronts ← Fast Non-Dominated Sort (C)
7:      Fronts_i ← Crowding Computation (Fronts_i) /* ∀i: i=0 to |Fronts|-1 */
8:      P ← 0, i ← 0 /* Updating the parent population */
9:      while |P| + |Fronts_i| < popSize do
10:         P ← P ∪ Fronts_i
11:         i ← i + 1
12:     end while
13:     P ← P ∪ Fronts_i [1:(popSize-|P|)]
14:     /* Generating offspring */
15:     for i = 1 to popSize do
16:         Q_i.m ← Apply Evolutionary Operators (P, crossoverProb, mutationProb)
17:         Q_i.τ ← Infer Phylogenetic Tree (Q_i.m, dataset)
18:         Q_i.scores ← Evaluate Solution (Q_i.τ, dataset)
19:     end for
20:     ParetoFront ← Update Pareto Front (P, Q, ParetoFront)
21: end while
```

This algorithm considers two populations P and Q of *popSize* individuals, where P refers to the parent population and Q is composed of offspring solutions, which are generated throughout the execution of NSGA-II by means of selection, crossover, and mutation operators. At each generation, these two populations are merged into an auxiliary population C, which contains *2*popSize* individuals (line 5 in Algorithm 10). This combined population is sorted according to the quality of its associated solutions in order to distinguish which ones should take the role of parents for the next offspring generation (lines 6-13). In a first step, NSGA-II applies a fast non-dominated sorting approach (line 6). As suggested in the explanation of the MOABC algorithm, the fast non-dominated sort assigns a rank value to each solution C_i in accordance with the number of individuals in C that dominates C_i, classifying the solutions into different Pareto fronts $Fronts_0$, $Fronts_1$, $Fronts_2$, ... where $Fronts_0$ contains the set of solutions with rank = 0 (non-dominated solutions), $Fronts_1$ is composed of solutions with rank = 1 (dominated by $Fronts_0$), $Fronts_2$ includes solutions with rank = 2 (dominated

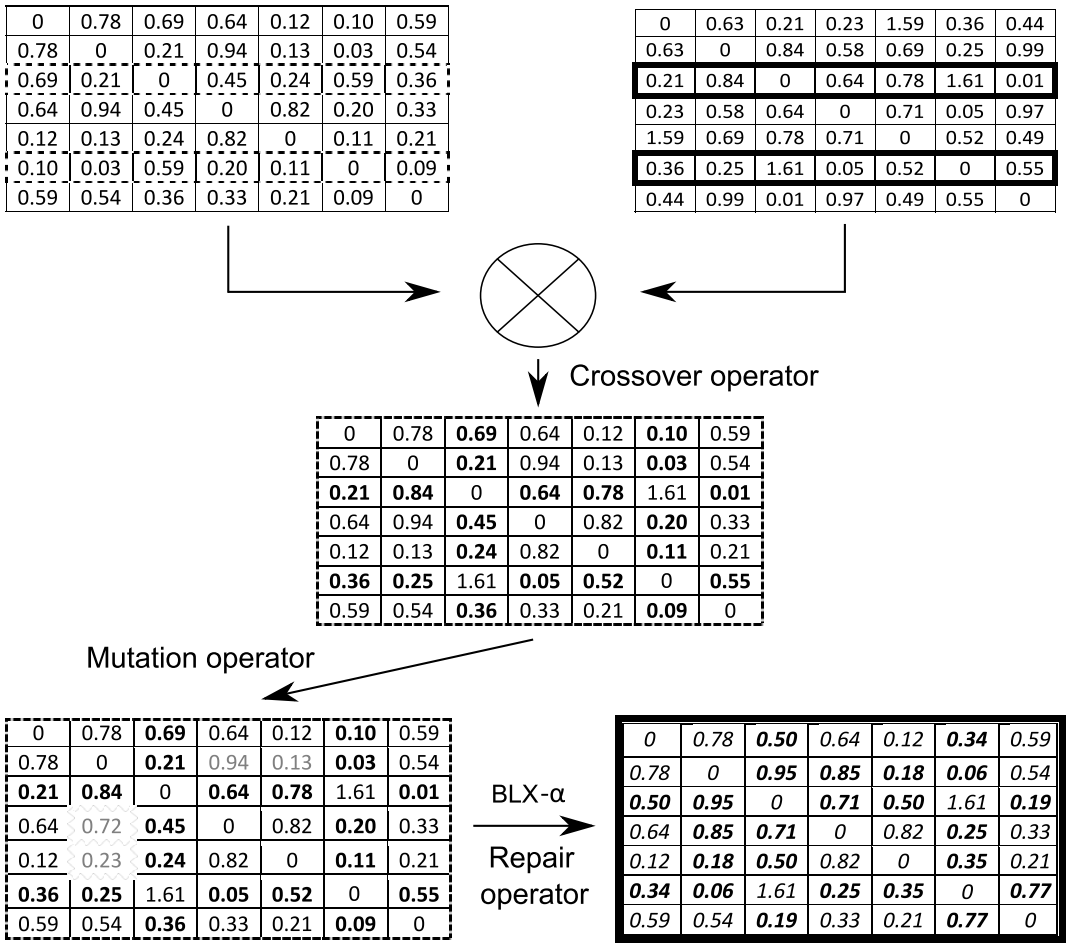


Fig. 5.2 Evolutionary operators: row-swapping uniform crossover, gamma distribution-based mutation and BLX-α repair operator

by $Fronts_0$ and $Fronts_1$) and so on. Given n objective functions and ϑ individuals to be sorted, the complexity of applying the fast non-dominated sorting approach is $O(n\vartheta^2)$.

Once rank values have been computed for each solution in C, an internal sorting of the obtained fronts must be carried out (line 7). NSGA-II performs this task by means of crowding distances, which provide an estimation of the density of points in the front surrounding a specific solution. With the aim of preserving diversity, the crowding distance gives more support to less crowded solutions, which will be preferred to other more crowded solutions. In this way, higher crowding distance values will be assigned to those less crowded points in the front. Sorting a front composed of ϑ individuals according to crowding distances requires at most $O(\vartheta log(\vartheta))$ operations.

After applying the fast non-dominated sort and crowding ordering, the parent population P is updated with the best *popSize* individuals identified through the processing of the combined population C (lines 8-13). From P, NSGA-II generates the solutions to be included in the offspring population Q by means of evolutionary operators (lines 14-19). In our adaptation of NSGA-II for phylogenetics, the following operators (represented in Figure 5.2) have been implemented:

1. For selection purposes, a binary tournament selection based on the crowded-comparison operator is used, as suggested in the original NSGA-II design. This crowded-comparison operator distinguishes the quality of competing solutions by using their rank and crowding values. When comparing two individuals with different ranks in the binary tournament, this operator will select as parent the one with the lowest rank value. If both individuals belong to the same front (same rank), the one with the highest crowding distance value will be chosen.

2. Regarding crossover, our implementation uses a uniform crossover operator which interchanges rows from two parent distance matrices to generate a new offspring matrix. This operator was chosen due to the significant results reported in [137], which were confirmed in our particular representation and MOEA set up in comparison to other operators, such as single point crossover and double point crossover.

3. The mutation operator modifies randomly selected entries in the offspring matrix attending to the gamma distribution of genetic rate variation observed in the input alignment (see Equation 5.2). As the resulting matrix must guarantee the symmetry restriction, BLX-α is also applied.

The distance matrix generated as a result of applying these evolutionary operators is mapped to the phylogenetic tree space by using the BIONJ tree-building method, recovering the corresponding phylogenetic tree, which is topologically optimized and evaluated afterwards. The offspring loop iterates until generating *popSize* new individuals, which are stored in the Q population. Finally, the *ParetoFront* structure is updated and a new generation takes place, repeating the tasks defined in the main loop of the algorithm until the stop criterion is satisfied.

5.2.4 Strength Pareto Evolutionary Algorithm 2 - SPEA2

SPEA2 is another well-known multiobjective metaheuristic proposed by Zitzler et al. [196]. As NSGA-II, SPEA2 is a population-based MOEA which manages a population P of individuals which evolve throughout the execution of the algorithm by means of evolutionary operators. SPEA2 design is supported by the concept of archive, an auxiliary set $\bar{P}$ which contains a representation of the most promising solutions found by the algorithm. The input parameters for this metaheuristic include the number of individuals in the population (*popSize*), the maximum number of solutions stored in the archive (*archiveSize*), the crossover probability (*crossoverProb*), and the mutation probability (*mutationProb*). SPEA2 algorithmic design is illustrated in Algorithm 11.

At each generation, the *popSize* individuals contained in the main population P and the *archiveSize* ones in the archive $\bar{P}$ are assessed by computing SPEA2 fitness values (line 4 in Algorithm 11). For this purpose, strength values are assigned to each individual in both P and $\bar{P}$ by verifying the number of solutions it dominates. For an individual i, its strength value $S(i)$ is defined as $S(i) = |j \in P + \bar{P} | i \succ j|$. These strength values are then used to compute raw fitness scores $R(i)$, defined as the summation of the strengths corresponding to the individuals which dominate the assessed

Algorithm 11 SPEA2 Pseudocode

Input: *maxEvaluations* (number of evaluations), *popSize* (number of individuals in the population), *archiveSize* (number of individuals in the archive), *crossoverProb* (crossover probability), *mutationProb* (mutation probability).
Output: *ParetoFront* (non-dominated solutions found by the algorithm).

```
 1: P̄, ParetoFront ← 0
 2: P ← Initialize Population (popSize, dataset)
 3: while ! stop criterion reached (maxEvaluations) do
 4:     P,P̄ ← Assign Fitness (P,P̄)
 5:     /* Environmental selection */
 6:     multiSet ← Fitness-Based Sorting (P ∪ P̄)
 7:     P̄ ← Copy Non-Dominated Individuals (multiSet)
 8:     if |P̄| < archiveSize then
 9:         P̄ ← Fill Archive (multiSet) [1:(archiveSize-|P̄|)]
10:     else
11:         while |P̄| > archiveSize do
12:             P̄ ← Truncate Individual (P̄)
13:         end while
14:     end if
15:     /* Generating offspring */
16:     for i = 1 to popSize do
17:         P_i.m ← Apply Evolutionary Operators (P̄, crossoverProb, mutationProb)
18:         P_i.τ ← Infer Phylogenetic Tree (P_i.m, dataset)
19:         P_i.scores ← Evaluate Solution (P_i.τ, dataset)
20:     end for
21:     ParetoFront ← Update Pareto Front (P, P̄, ParetoFront)
22: end while
```

solution: $R(i) = \sum_{j \in P+\bar{P}, j \succ i} S(j)$. This fitness mechanism is complemented by adding density information to raw fitness in the following way. For each solution, it is calculated the distance (in terms of objective function scores) to all individuals in P and $\bar{P}$, computing afterwards the inverse of the distance to its k-th nearest neighbour δ^k, where $k = \sqrt{popSize + archiveSize}$. Thereby, the density value $D(i)$ for an individual i is: $D(i) = \frac{1}{\delta_i^k + 2}$. The final fitness score of i is defined as $F(i) = R(i) + D(i)$. By using this fitness assignment procedure, the multiobjective quality of the solutions handled by the algorithm can be distinguished, taking into account that non-dominated solutions are associated to fitness values lower than one. The complexity of the raw fitness and density estimation methods is approximated as $O(\vartheta^2)$ and $O(\vartheta^2 log(\vartheta))$, respectively, where ϑ=*popSize*+*archiveSize*.

After computing fitness values, the algorithm undertakes the environmental selection of the best current solutions (lines 5-14). In a first step, the solutions contained in both P and $\bar{P}$ are conveniently stored and ordered in a *multiSet* structure attending to their fitness values (line 6). Afterwards, the archive $\bar{P}$ is updated by choosing all the non-dominated solutions included in *multiSet* from both P and $\bar{P}$ (line 7). If the number of solutions in the updated $\bar{P}$ is lower than *archiveSize*, the best *archiveSize*-$|\bar{P}|$ dominated solutions will be used to fill the archive (line 9). On the other hand, if the number of solutions copied to $\bar{P}$ is larger than *archiveSize*, a truncation operator which deletes the individual with the minimum distance to another individual is applied until $|\bar{P}|$ fits *archiveSize* (lines 11-13). The computational complexity of the archive update step is dominated by the truncation procedure, which shows a time cost of $O(\vartheta^3)$.

On the basis of the updated archive, new offspring solutions are generated and stored in P by performing selection, crossover, and mutation (lines 15-20). The adaptation of SPEA2 to phyloge-

netic inference includes the same crossover, mutation, and repair operators considered in NSGA-II. In this case, binary tournament selections are carried out according to the SPEA2 fitness values of the competing solutions. The matrices generated by the evolutionary operators will be processed to infer the corresponding phylogenetic trees via BIONJ and topological optimization, computing parsimony and likelihood scores over the resulting phylogenies for assessment purposes. As usual, the non-dominated solutions found in the current generation are used to update the *ParetoFront* structure, repeating the main loop of SPEA2 until reaching the stop criterion.

5.3 Indicator-Based Approaches

The increasingly popular indicator-based approaches represent the second class of algorithms considered in this Thesis. As remarked in Chapter 4, the assessment of multiobjective metaheuristics is usually conducted by means of quality indicators which provide a real number measurement of their quality. Indicator-based proposals go a step further in the development of multiobjective designs by including fitness assignment and selection mechanisms which are built upon the information provided by quality indicators. In this way, the optimization goal is to generate a set of solutions that optimize the considered performance measurement metric. Among the different quality indicators proposed in the literature, hypervolume represents one of the most reliable and popular metrics. Let X be the outcome of a multiobjective algorithm, defined as a set of points in the objective space Z=$\Re^n$. Formally, the hypervolume $I_H(X)$ measures the n-dimensional volume (area for a bi-dimensional MOP) of the objective space (with regard to a reference point Z_{ref}) which is covered by at least one point $s \in X$ [14]. The implementations of indicator-based fitness procedures in this research consider the application of the I_{HD} indicator [195], a hypervolume-based metric which compares two sets X,Y by computing the volume of the objective space dominated by Y but not by X:

$$I_{HD}(X,Y) = \begin{cases} I_H(Y) - I_H(X) & \text{if } \forall s \in Y, \exists s' \in X : s' \succ s, \\ I_H(X+Y) - I_H(X) & \text{otherwise.} \end{cases} \tag{5.5}$$

By considering unary sets, I_{HD} can be used to make pairwise comparisons of solutions in the fitness assignment steps. In this Thesis, we analyze two indicator-based multiobjective designs: a swarm intelligence algorithm (IMOBA), and an evolutionary algorithm (IBEA).

5.3.1 Indicator-Based Multiobjective Bat Algorithm - IMOBA

The hunting strategies of bats in nature are built upon their echolocation capabilities. Bats are able to emit sound pulses and listen to the echo that bounces back from the objects surrounding them in the environment. When looking for prey, bats fly with a certain velocity, sending sound pulses on a given frequency range with a varying emission rate and loudness. When prey is detected, the rate of pulse emission progressively grows and the loudness decreases as bats come closer to the target. In this

Algorithm 12 IMOBA Pseudocode

Input: *maxEvaluations* (number of evaluations), *popSize* (number of individuals in the population), *mutationProb* (mutation probability), $[f_{min}, f_{max}]$ (frequency range), A_0 (initial loudness), α (loudness update factor), r_{max} (maximum pulse emission rate), γ (pulse update factor), κ (scaling factor), Z_{ref} (hypervolume reference point).
Output: *ParetoFront* (non-dominated solutions found by the algorithm).

1: *ParetoFront* ← 0
2: P ← Initialize Population (*popSize*, *dataset*)
3: P ← Initialize Bat Parameters (*popSize*, f_{max}, f_{min}, A_0, r_{max})
4: **while** ! stop criterion reached (*maxEvaluations*) **do**
5: P_i ← Indicator-Based Fitness Assignment (κ, Z_{ref}) /* $\forall i$: i=1 to $|P|$ */
6: P ← Environmental Selection (*popSize*, κ) /* *Saving the best popSize bats* */
7: /* *Bat movement step* */
8: **for** i = 1 to *popSize* **do**
9: $P'_i.m$ ← Move Bat (P_i, f_{max}, f_{min})
10: $P'_i.\tau$ ← Infer Phylogenetic Tree ($P'_i.m$, *dataset*)
11: $P'_i.scores$ ← Evaluate Solution ($P'_i.\tau$, *dataset*)
12: **end for**
13: $P \leftarrow P \cup P'$
14: P_i ← Indicator-Based Fitness Assignment (κ, Z_{ref}) /* $\forall i$: i=1 to $|P|$ */
15: P ← Environmental Selection (*popSize*, κ) /* *Saving the best popSize bats* */
16: selectionArray ← Compute Selection Probabilities (P, *popSize*)
17: /* *Bat search step* */
18: **for** i = 1 to *popSize* **do**
19: **if** rand $> P_i.r$ **then**
20: $P'_i.m$ ← Generate Exploitation Bat (selectionArray, *mutationProb*)
21: **else**
22: $P'_i.m$ ← Generate Exploration Bat (*mutationProb*)
23: **end if**
24: $P'_i.\tau$ ← Infer Phylogenetic Tree ($P'_i.m$, *dataset*)
25: $P'_i.scores$ ← Evaluate Solution ($P'_i.\tau$, *dataset*)
26: **end for**
27: $P \leftarrow P \cup P'$
28: $P_i.A$ ← Update Loudness (α, $P_i.A$) /* $\forall i$: i=1 to $|P|$ */
29: $P_i.r$ ← Update Emission Rate (γ, $P_i.r_{max}$) /* $\forall i$: i=1 to $|P|$ */
30: *ParetoFront* ← Update Pareto Front (P, *ParetoFront*)
31: **end while**

way, bats can process the distance, orientation, and moving speed of the prey, adapting the emission of sound pulses according to the proximity to the target to ensure the success of the hunting.

This behaviour can be modelled to address optimization problems, as proposed in BA, a swarm intelligence algorithm reported by Yang [191]. IMOBA is proposed in this Thesis as an indicator-based multiobjective adaptation of this novel algorithm. It applies bat hunting strategies in combination with indicator-based fitness assignment procedures to carry out the search for high-quality approximation sets. The input parameters for this metaheuristic are related to the modelling of sound pulse emissions and bat movements, including a pulse frequency range ($[f_{min}, f_{max}]$), initial pulse loudness (A_0), a loudness update factor (α), the maximum pulse emission rate (r_{max}), and a pulse update factor (γ). Other parameters specify the number of bats in the swarm (*popSize*), the mutation probability (*mutationProb*), a scaling factor used in indicator-based fitness computations (κ), and the I_{HD} reference point (Z_{ref}). IMOBA pseudocode is shown in Algorithm 12.

At the beginning, each bat in the swarm P is initialized (lines 2 and 3 in Algorithm 12) by assigning a starter position given by a distance matrix m_0, a velocity matrix $v_0 = 0$, a pulse frequency f_0 in the range $[f_{min}, f_{max}]$, a pulse emission rate $r_0 \in [0, r_{max}]$, and loudness A_0. IMOBA main loop involves the following steps. Firstly, the environmental selection of the *popSize* most promising bats

in P is carried out (lines 5-6). To this end, IMOBA applies the indicator-based fitness assignment and selection procedures proposed in IBEA [195]. In short, I_{HD} values are computed for each bat in P, calculating fitness values for a bat P_i by summing up its computed I_{HD} scores with respect to each remaining bat P_j. The environmental selection is performed to retain the most satisfying *popSize* bats in P by iteratively removing the bat in the swarm which shows the smallest fitness value. Full details on the indicator-based fitness assignment and environmental selection are provided in the explanation of IBEA in the next section. Next, bats in P are used to compute new candidate solutions in the movement step (lines 7-12). Let P_i be a bat with velocity $P_i.v$ and frequency $P_i.f$ associated to a distance matrix $P_i.m$. A new position (matrix) for P_i is calculated by using the following expressions:

$$P_i.f = f_{\min} + (f_{\max} - f_{\min})\beta, \tag{5.6}$$

$$P_i.v[i,j] = P_i.v[i,j] + (P_i.m[i,j] - P_{rand}.m[i,j])P_i.f, \tag{5.7}$$

$$P_i.m[i,j] = P_i.m[i,j] + P_i.v[i,j], \tag{5.8}$$

where $\beta \in [0,1]$ is a random number taken from a uniform distribution and P_{rand} refers to a randomly chosen bat in P. This operation is complemented by using BLX-α to preserve the symmetry property in the matrix, obtaining and evaluating afterwards the phylogenetic tree. The resulting solutions are stored in an auxiliary swarm P' which will merge with the original P, in such a way that a new environmental selection will take place to maintain the best *popSize* bats (lines 13-15). As an additional task, a selection probability array is computed by considering the position of each bat in P after sorting them according to their fitness values (line 16).

This selection array gives support to the second main step in the algorithm (lines 17-26), which involves a exploitation-exploration procedure inspired by the variations observed in pulse emission rates and loudness when bats search for prey. The decision on performing an exploitation or exploration step depends on the current emission rate $P_i.r$ of the examined bat (see line 19, where rand refers to a random number in the range [0,1]). In the exploitation branch, a new candidate solution P_{new} is generated by initializing its bat parameters and computing a distance matrix as follows:

$$P_{new}.m[i,j] = P_{sel}.m[i,j] + \varepsilon \langle A \rangle, \tag{5.9}$$

where P_{sel} refers to a solution in P chosen by means of the selection array, ε is a random number in the interval $[-1,1]$, and $\langle A \rangle$ is the average loudness of all the bats in P. In this expression, the entries to modify are chosen according to the mutation rate parameter. BLX-α is also used for symmetrization purposes. On the other hand, in the exploration branch, a new distance matrix $P_{new}.m$ is computed by randomly selecting a starter phylogeny from the tree repository, applying topological optimization to ensure that the new solution is able to compete with the ones previously found by the algorithm. The results of this second main step are stored in P' and incorporated into the swarm P (line 27).

The final steps of each generation involve the update of loudness and emission rates (lines 28 and 29). These operations are formulated for a bat P_i as: $P_i.A = \alpha P_i.A$ and $P_i.r = P_i.r_{max}[1 - \exp(-\gamma t)]$, where α and γ are constant factors and t identifies the current generation. Afterwards, the *ParetoFront* structure is updated and the next generation takes place.

5.3.2 Indicator-Based Evolutionary Algorithm - IBEA

Originally proposed by Zitzler and Künzli [195], this indicator-based MOEA involves the evolution of a population P whose individuals are assessed by making pairwise comparisons in terms of the chosen quality indicator. Algorithm 13 summarizes IBEA algorithmic design, whose adaptive implementation has been considered in this research. The input parameters for IBEA include the number of individuals in the population (*popSize*), crossover and mutation probabilities (*crossoverProb*, *mutationProb*), a scaling factor used in fitness computations (κ), and the I_{HD} reference point (Z_{ref}).

The main loop of IBEA operates over the population P by applying an indicator-based environmental selection of the best individuals according to how useful they are attending to the quality indicator. The selected individuals will be used to generate new offspring solutions by means of evolutionary operators. Each generation involves the following steps. Firstly, the current state of each individual in the population is assessed according to the considered quality indicator. To this end, the values $I_{HD}(P_i, P_j)$ are computed for each pair of individuals P_i, P_j (lines 6-10 in Algorithm 13) by considering their normalized objective function scores (parsimony and likelihood) in the interval [0,1]. Afterwards, the IBEA fitness assignment procedure takes place (line 11), with the aim of quantifying, for each individual P_i, the loss in quality observed if P_i is removed from the population. The fitness assignment for an individual P_i is carried out by summing up its I_{HD} values with respect to each remaining individual P_j as: $P_i.Fitness = \sum_{P_j \in P \setminus \{P_i\}} -e^{-I_{HD}(\{P_j\},\{P_i\})/c \cdot \kappa}$. In this expression, c denotes the maximum absolute indicator value, which is included in the adaptive version of IBEA to avoid the effect of widely spread indicator scores.

The computed fitness values are examined in the next step of the algorithm, which applies an environmental selection procedure over the population (lines 12-16) to keep the most promising *popSize* individuals. This environmental selection proceeds by identifying iteratively the individual P_{worst} which shows the smallest (worst) fitness value, removing it from P and updating the fitness values of the remaining individuals in the following way: $P_i.Fitness = P_i.Fitness + e^{-I_{HD}(\{P_{worst}\},\{P_i\})/c \cdot \kappa}$. This procedure is repeated until the size of P fits the parameter *popSize*. After performing this step, the algorithm defines a mating pool P' used in the generation of new offspring solutions (lines 17-22).

In this adaptation of IBEA to phylogeny reconstruction, the following evolutionary operators are included. Firstly, parent selection is carried out by performing binary tournaments in accordance with the indicator-based fitness values calculated by the algorithm. Secondly, crossover, mutation, and repair mechanisms are implemented by using the same operators specified in the adaptations of NSGA-II and SPEA2. From the resulting distance matrices, the corresponding phylogenetic trees are inferred, optimized, and evaluated according to the objective functions. Afterwards, P' is integrated

Algorithm 13 IBEA Pseudocode

Input: *maxEvaluations* (number of evaluations), *popSize* (number of individuals in the population), *crossoverProb* (crossover probability), *mutationProb* (mutation probability), κ (scaling factor), Z_{ref} (hypervolume reference point).
Output: *ParetoFront* (non-dominated solutions found by the algorithm).

```
1:  P', ParetoFront ← 0
2:  P ← Initialize Population (popSize, dataset)
3:  while ! stop criterion reached (maxEvaluations) do
4:      /* Indicator-based fitness assignment */
5:      P ← Scale Objective Values (bounds(P)) /* To the interval [0,1] */
6:      for i = 1 to |P| do
7:          for j = 1 to |P| do
8:              indicatorValues[i][j] ← Calculate I_HD Values (P_i, P_j, Z_ref)
9:          end for
10:     end for
11:     P_i.Fitness ← Assign Fitness (indicatorValues, κ) /* ∀i: i=1 to |P| */
12:     /* Environmental selection */
13:     while |P| > popSize do
14:         P_worst ← Remove Individual with Smallest Fitness (P)
15:         P_i.Fitness ← Update Fitness (indicatorValues, P_worst, κ) /* ∀i: i=1 to |P| */
16:     end while
17:     /* Generating offspring */
18:     for i = 1 to popSize do
19:         P'_i.m ← Apply Evolutionary Operators (P, crossoverProb, mutationProb)
20:         P'_i.τ ← Infer Phylogenetic Tree (P'_i.m, dataset)
21:         P'_i.scores ← Evaluate Solution (P'_i.τ, dataset)
22:     end for
23:     P ← P ∪ P'
24:     ParetoFront ← Update Pareto Front (P, ParetoFront)
25: end while
```

into the original population (line 23), starting a new generation after performing the update of the *ParetoFront* structure. These steps take place iteratively until the stop criterion is satisfied.

5.4 Summary

A full description of the multiobjective metaheuristics examined to tackle the phylogeny inference problem has been provided in this chapter. First of all, some common implementation features have been detailed, defining a distance-based search methodology in which solutions are encoded in terms of distance matrices. The main goal behind this individual representation is to perform phylogenetic searches over a matrix-shaped space, applying floating-point evolutionary operators over genetic distances to generate new solutions that are later mapped to the phylogenetic tree space. On the basis of this methodology, up to six multiobjective algorithms have been applied to solve the problem.

The first main class of multiobjective designs studied in this Thesis comprises those algorithms which apply the Pareto dominance concept in the fitness assignment steps, the dominance-based multiobjective approaches. Four dominance-based algorithms have been explained: two multiobjective adaptations of novel swarm intelligence algorithms and two reference MOEAs which are widely-used in the field of multiobjective optimization. Regarding the swarm intelligence proposals, two designs have been considered. Firstly, a multiobjective algorithm inspired in the collective behaviour of honey bees, named as MOABC, takes advantage of the exploration-exploitation capabilities of these social insects when searching for food sources, identifying three main types of bees in the hive:

employed bees, onlooker bees, and scout bees. Secondly, the bioluminescence properties of fireflies have been modelled to define a multiobjective swarm intelligence algorithm, MO-FA. This algorithm undertakes the search for satisfying Pareto approximation sets by identifying the attractiveness of a solution with its quality, in such a way that the firefly swarm will move towards the positions of the best solutions according to the factors that govern the movement of fireflies in nature. On the other hand, two well-known MOEAs, NSGA-II and SPEA2, have been applied to verify the quality of these standard multiobjective metaheuristics when addressing the problem studied in this Thesis.

The second class of multiobjective designs undertakes the processing of the search space in accordance with the guidance support provided by a quality indicator. These approaches are known as indicator-based algorithms. Two indicator-based metaheuristics have been applied in this research: IMOBA and IBEA. IMOBA exploits the echolocation capabilities shown by bats when hunting for prey, extending the recent BA proposal to tackle MOPs. The proposed IMOBA defines exploration-exploitation mechanisms based on the movement of bats and the emission of sound pulses to carry out the search for high-quality phylogenies attending to several factors, such as velocity, frequency, loudness, and pulse emission rates. On the other hand, IBEA is a representative indicator-based MOEA proposed with the aim of providing a framework for the integration of quality indicator computations into multiobjective search. In summary, six multiobjective bioinspired metaheuristics have been detailed, explaining their algorithmic designs and their adaptation to phylogenetic inference. The next chapter examines the success of these approaches when dealing with this problem, assessing the obtained results in terms of multiobjective and biological quality.

Chapter 6

Multiobjective and Biological Results

This chapter discusses the multiobjective and biological relevance of the results attained by the examined multiobjective metaheuristics when solving the phylogenetic inference problem. For this purpose, some initial experimental considerations are introduced, providing details on the statistical model selection, quality indicator computations, parametric configuration studies, and justification on the choice of the distance-based phylogenetic encoding and the BIONJ tree-building method used in the implemented search engines. Later, the chapter is focused on carrying out a comparative study of multiobjective performance, examining the quality of the Pareto approximation sets obtained by the algorithms under the statistical methodology followed in this research. Starting from the evaluation of dominance-based and indicator-based approaches separately, a direct comparison between these two main lines of multiobjective designs will be performed afterwards, in order to identify the best metaheuristic from a multiobjective performance perspective. Finally, the biological assessment of the inferred phylogenetic trees is conducted, making comparisons with the state-of-the-art and examining the biological implications of multiobjective optimization.

6.1 Initial Considerations

This section is aimed at reporting some initial experimental considerations and additional details on the methodology followed to conduct the evaluation of the results reported by the implemented metaheuristics. Firstly, the results of performing the statistical selection of evolutionary models for each dataset considered in this research are examined, identifying the best probabilistic framework to conduct our phylogenetic analyses. Secondly, details on the computation of the metrics used for multiobjective comparison purposes are provided, giving account of the parametric studies performed to carry out the configuration of input parameters for each algorithm. Finally, some of the most important metaheuristic design choices assumed in this Thesis, such as the use of distance matrices to encode candidate solutions and the integration of BIONJ as tree-building method, are discussed.

As previously reported in Chapter 4, the experiments carried out to verify the success of the examined metaheuristics in addressing real phylogenetic reconstructions were performed on a hard-

ware setup containing an AMD Opteron Magny-Cours 6174 processor at 2.2 GHz with 12MB L3 cache and 16GB DDR3 RAM, running Scientific Linux 6.6 as operating system. The methods tested in this chapter (including the six implemented metaheuristics and biological methods from other authors) were compiled by using GCC with the -O3 optimization flag. Our experiments involve 31 independent executions per dataset of the considered algorithms, in order to report statistically consistent conclusions based on the median results observed throughout our experimentation. To provide more statistical robustness to this study, the results samples obtained in these experiments (i.e. hypervolume results) were analyzed under the statistical methodology described in Chapter 4 - Section 4.5, along with the application of biological phylogenetic testing procedures (Section 4.3.3) to verify the quality of our inferred trees with regard to other authors' proposals.

6.1.1 Data Sets and Evolutionary Model Selection

In this chapter, the assessment of multiobjective and biological performance is carried out by analyzing the quality of the outcomes generated by the implemented metaheuristics on six real biological data sets: *rbcL_55* (55 nucleotide sequences with 1314 sites per sequence), *mtDNA_186* (186 nucleotide sequences with 16608 sites per sequence), *HIV1_192* (192 nucleotides sequences with 817 sites per sequence), *RDPII_218* (218 nucleotide sequences with 4182 sites per sequence), *S1482_346* (346 nucleotide sequences with 897 sites per sequence), and *ZILLA_500* (500 nucleotide sequences with 759 sites per sequence).

In order to conduct successful phylogenetic analyses on these six data sets under model-based optimality criteria (i.e. likelihood), statistical model selection procedures must be carried out. The main idea lies on identifying the model of nucleotide evolution which fits best the evolutionary process observed for a given dataset. With this purpose, these data sets have been analyzed by using the jModelTest tool [48] to evaluate the accuracy of the models and determine the best model parameter configuration (such as the rates of transitions/transversions and nucleotide base frequencies) according to the characteristics of the input alignments. We are interested in studying three widely-used evolutionary models: $HKY85+\Gamma$ [78], $TN93+\Gamma$ [178] and $GTR+\Gamma$ [145], considering the results reported by jModelTest for the AIC and BIC tests.

Table 6.1 presents the results of the statistical model selection for each dataset. This Table shows the likelihood of the phylogeny computed by jModelTest to conduct model comparisons (L), the number of parameters in the evaluated model (K, which includes the number of estimated tree branches), and the calculated AIC/BIC scores attending to Equations 4.8 and 4.9 (which are to be minimized). According to the obtained results, $GTR+\Gamma$ represents the most accurate model choice in almost all the considered data sets. Although this model is surpassed in terms of AIC and BIC scores by $TN93+\Gamma$ on *mtDNA_186*, the likelihood result of $GTR+\Gamma$ for this alignment improves the one reported by $TN93+\Gamma$ (-39880.5 vs. -39882.7). On the other hand, $HKY85+\Gamma$ is outperformed by $GTR+\Gamma$ and $TN93+\Gamma$ in almost all the AIC/BIC scenarios. On the basis of these results, $GTR+\Gamma$ was chosen as the fittest probabilistic model to carry out our experimentation.

Table 6.1 AIC - BIC statistical model selection tests

	rbcL_55				*mtDNA_186*			
Model	L	K	AIC	BIC	L	K	AIC	BIC
$GTR+\Gamma$	-21834.9	117	**43903**	**44509**	-39880.5	379	80519	83444
$TN93+\Gamma$	-21853.9	114	43936	44526	-39882.7	376	**80517**	**83419**
$HKY85+\Gamma$	-21897.0	113	44020	44605	-39906.4	375	80563	83457
	HIV1_192				*RDPII_218*			
$GTR+\Gamma$	-21851.4	391	**44485**	**46325**	-134394.8	443	**269675**	**272483**
$TN93+\Gamma$	-21941.5	388	44659	46485	-134489.5	440	269859	272648
$HKY85+\Gamma$	-21943.6	387	44661	46482	-134882.2	439	270642	273425
	S1482_346				*ZILLA_500*			
$GTR+\Gamma$	-9903.6	698	**21203**	**24553**	-80775.6	1007	**163565**	**168230**
$TN93+\Gamma$	-10084.5	695	21559	24894	-81160.5	1004	164329	168980
$HKY85+\Gamma$	-10087.6	694	21563	24894	-81161.9	1003	164330	168976

6.1.2 Multiobjective Performance Metrics

In the research undertaken in this Thesis, the evaluation of multiobjective performance has been performed by assessing the quality of the inferred Pareto sets attending to different perspectives, such as convergence and solution diversity. As stated in Chapter 4 - Section 4.3.1, three multiobjective quality indicators have been used for this purpose: hypervolume [14], coverage relation or set coverage [194], and spacing [183]. These metrics have been applied to measure multiobjective performance according to the following guidelines.

For each metaheuristic, the hypervolume of the covered objective space has been computed by normalizing the objective scores of the inferred non-dominated solutions to the scale [0,1]. This idea aims to avoid the influence of the different score scales shown by parsimony (integer) and likelihood (float) in hypervolume values. The ideal and nadir points used to calculate this metric can be found in Table 6.2, which shows the estimated parsimony and likelihood scores of these points for each dataset. This estimation was experimentally conducted by considering the best objective function scores (for the ideal point) and the worst ones (for the nadir point) obtained by the metaheuristics under study, adding/subtracting a certain percentage of their values (2% for parsimony and 8% for likelihood). For comparison purposes, the hypervolume indicator has been configured to measure the area covered by each normalized front with regard to the reference point $Z_{ref} = (1,1)$, taking both objectives to be minimized by considering positive likelihood values.

As a result of performing 31 independent runs per experiment, 31 hypervolume samples per dataset have been obtained for each metaheuristic. In order to summarize the information retrieved by this quality indicator, median hypervolume values and their interquartile ranges will be reported throughout this chapter. For the sake of simplicity, the computation of the remaining quality indicators (set coverage and spacing) has been conducted by considering the Pareto fronts from the median hypervolume execution reported by each algorithm. In this way, we can state that the pairwise comparison of Pareto fronts between algorithms and the measurement of point distribution uniformity have been performed over representative samples of the outcomes generated by each metaheuristic.

Table 6.2 Hypervolume normalization points

	Ideal point		Nadir point	
Dataset	Parsimony	Likelihood	Parsimony	Likelihood
rbcL_55	4774	-21569.69	5279	-23551.42
mtDNA_186	2376	-39272.20	2656	-43923.99
HIV1_192	5981	-21522.56	11333	-28026.41
RDPII_218	40658	-132739.90	45841	-147224.59
S1482_346	2020	-9570.64	2304	-10593.35
ZILLA_500	15893	-79798.03	17588	-87876.39

6.1.3 Input Settings Configuration

In this research, the phylogenetic inference problem has been tackled by using a variety of multiobjective metaheuristics. Their algorithmic designs are built upon different principles (swarm intelligence vs. evolutionary algorithms, dominance-based designs vs. indicator-based approaches, etc.). As a result, we are dealing with algorithms of different nature whose input parameters must be configured properly to achieve satisfying solutions to this problem. In order to find the best configuration of input parameters for each algorithm, we have examined the Pareto approximation sets generated from different configurations, evaluating each outcome by using the hypervolume metrics. More specifically, different values uniformly distributed in the parameter range were verified, in such a way that the parameters with the greatest influence on hypervolume were the first to be fixed and so on. Table 6.3 reports the input parameter values studied for each metaheuristic. In this table, the first column refers to the examined parameter, while the next five columns show the values checked in our configuration experiments. The last column reports the input parameter value which showed the best overall behaviour in terms of improved hypervolume results, representing the optimal setting value found in the experimentation associated to this parametric study.

Focusing on each particular metaheuristic, the optimal settings found for MOABC involve a population size of 100 bees, a mutation rate of 5%, and a limit of 25 generations allowed before a solution is replaced by the one retrieved by a scout search. MO-FA considers a population of 125 fireflies, an attractiveness factor of 1, an environmental light absorption coefficient of 0.5, and a randomization factor of 0.05. Regarding NSGA-II, SPEA2 and IBEA, the best settings found for these three MOEAs include a population size of 100 individuals, a crossover probability of 70%, and a mutation rate of 5%. For the case of SPEA2, the size of the archive is set to 75 individuals. The configuration of the I_{HD} quality indicator in IBEA uses the reference point (2,2), along with a scaling factor in the fitness assignment procedure of 0.05. Finally, IMOBA represents a parameter-intensive metaheuristic, whose optimal configuration for this problem considers a population of 100 bats, a mutation probability of 5%, frequencies in the range [0,1], a loudness update factor of 0.9, a maximum pulse emission rate of 0.5, and a pulse update factor of 0.3. The initial loudness in IMOBA takes a specific value for each dataset which is related to the addressed problem, the shape parameter

Table 6.3 Parametric studies to configure the assessed metaheuristics

Stop criterion = maximum number of evaluations: 10000						
Dominance-based approaches						
MOABC	Tested values					**Best value**
Population size (*popSize*)	25	50	75	100	125	**100**
Mutation rate (*mutationProb*)	5%	10%	15%	20%	25%	**5%**
Limit control parameter (*limit*)	5	15	25	35	45	**25**
MO-FA	Tested values					**Best value**
Population size (*popSize*)	25	50	75	100	125	**125**
Attractiveness factor (β_0)	0.1	0.25	0.5	0.75	1	**1**
Absorption coefficient (γ)	0.1	0.25	0.5	0.75	1	**0.5**
Randomization factor (α)	0.05	0.1	0.15	0.2	0.25	**0.05**
NSGA-II	Tested values					**Best value**
Population size (*popSize*)	25	50	75	100	125	**100**
Crossover rate (*crossoverProb*)	50%	60%	70%	80%	90%	**70%**
Mutation rate (*mutationProb*)	5%	10%	15%	20%	25%	**5%**
SPEA2	Tested values					**Best value**
Population size (*popSize*)	25	50	75	100	125	**100**
Archive size (*archiveSize*)	10	25	50	75	100	**75**
Crossover rate (*crossoverProb*)	50%	60%	70%	80%	90%	**70%**
Mutation rate (*mutationProb*)	5%	10%	15%	20%	25%	**5%**
Indicator-based approaches						
IMOBA	Tested values					**Best value**
Population size (*popSize*)	25	50	75	100	125	**100**
Mutation rate (*mutationProb*)	5%	10%	15%	20%	25%	**5%**
Frequency range ($[f_{min}, f_{max}]$)	[0,1]	[0,50]	[0,100]	[1,2]	[1,50]	**[0,1]**
Initial loudness (A_0)	Gamma shape parameter					**Gamma**
Loudness update factor (α)	0.25	0.5	0.75	0.9	1	**0.9**
Maximum pulse emission rate (r_{max})	0.25	0.5	0.75	0.9	1	**0.5**
Pulse update factor (γ)	0.15	0.3	0.45	0.6	0.75	**0.3**
Scaling factor (κ)	0.05	0.075	0.1	0.25	0.5	**0.05**
I_{HD} reference point (Z_{ref})	(1.1,1.1)	(1.25,1.25)	(1.5,1.5)	(1.75,1.75)	(2,2)	**(2,2)**
IBEA	Tested values					**Best value**
Population size (*popSize*)	25	50	75	100	125	**100**
Crossover rate (*crossoverProb*)	50%	60%	70%	80%	90%	**70%**
Mutation rate (*mutationProb*)	5%	10%	15%	20%	25%	**5%**
Scaling factor (κ)	0.05	0.075	0.1	0.25	0.5	**0.05**
I_{HD} reference point (Z_{ref})	(1.1,1.1)	(1.25,1.25)	(1.5,1.5)	(1.75,1.75)	(2,2)	**(2,2)**

of the gamma distribution observed in genetic rate variation [105]. I_{HD} computations in IMOBA also consider a reference point (2,2) and a scaling factor of 0.05.

6.1.4 Discussing the Distance-based Representation

The adaptation of the examined multiobjective metaheuristics to phylogenetic inference considers a distance-based search methodology in which candidate solutions are encoded in terms of distance matrices. The search is conducted by applying evolutionary operators over the matrix search space and inferring the corresponding topologies by using a distance-based tree-building method. In order to justify the relevance of applying distance-based search engines to phylogenetics in a multiobjective

Table 6.4 Comparison of MOABC implementations under hypervolume

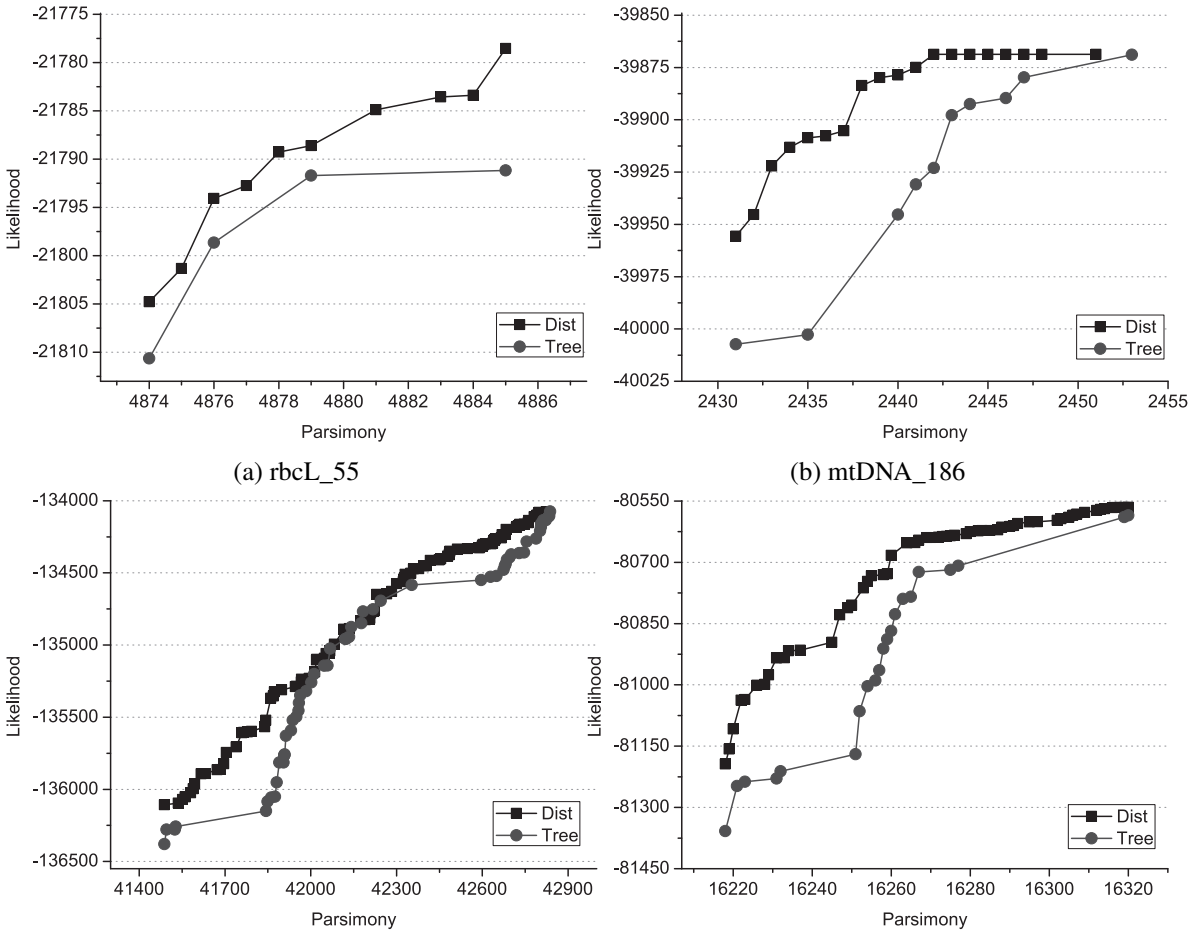
Hypervolume					
	Dist		Tree		Stat.
Dataset	Median	IQR	Median	IQR	Sign.?
rbcL_55	**71.7323**	0.0180	71.4298	0.1305	✓
mtDNA_186	**70.0205**	0.0140	69.9402	0.0103	✓
RDPII_218	**74.6408**	0.0500	74.2332	0.1212	✓
ZILLA_500	**73.0155**	0.0120	72.7041	0.1472	✓

Table 6.5 Comparison of MOABC implementations under set coverage and spacing

	Coverage Relation		Spacing	
Dataset	SC(Dist,Tree)	SC(Tree,Dist)	Dist	Tree
rbcL_55	**100.0000**	0.0000	**0.0867**	0.0952
mtDNA_186	**100.0000**	0.0000	**0.0793**	0.1684
RDPII_218	**89.4737**	6.1225	**0.0168**	0.0448
ZILLA_500	**100.0000**	0.0000	**0.0309**	0.1201

context, we introduce a comparison of the results reported by this approach with regard to classic tree-based implementations which operate directly over the tree search space. As pointed out in Chapter 2, a tree-based engine generates new solutions by means of topological rearrangements (NNI, SPR ...) which are applied over a set of starter topologies, improving them throughout the execution of the algorithm. This comparison between search methodologies has been carried out by using distance-based and tree-based implementations of the MOABC algorithm. In short, the tree-based implementation of this metaheuristic involves the application of NNI moves to model the exploitation steps undertaken by employed and onlooker bees, including the same topological optimization procedure (based on SPR, NNI, and gradient search) in the scout bees step.

Experimentation with these two implementations has been conducted on four real benchmark instances: *rbcL_55*, *mtDNA_186*, *RDPII_218*, and *ZILLA_500*. The generated outcomes have been analyzed under hypervolume, set coverage, and spacing, in order to find out what design leads to the best multiobjective behaviour. The results of assessing these implementations according to the hypervolume indicator are provided in Table 6.4. This table shows the median hypervolume values obtained by the distance-based (named as Dist) and tree-based (Tree) implementations of MOABC (columns 2 and 4, respectively), along with the observed interquartile ranges (IQR, columns 3 and 5). Column 6 points out the results of conducting our statistical testing methodology over the hypervolume samples generated by each implementation, reporting if statistically significant differences were found (✓) or not (×). In addition, Table 6.5 introduces the comparison of the median hypervolume Pareto fronts (represented in Figure 6.1) according to the coverage relation (columns 2 and 3, where SC(Dist,Tree) refers to the percentage of solutions in Tree which are covered by the solutions in Dist and SC(Tree,Dist) the points in Dist which are covered by Tree) and spacing (columns 4 and 5).



(a) rbcL_55 (b) mtDNA_186

(c) RDPII_218 (d) ZILLA_500

Fig. 6.1 Pareto fronts achieved by the distance-based and tree-based versions of MOABC

This comparison gives account of the relevant results achieved by the proposed distance-based methodology, outperforming the traditional tree-based search approach under these three quality indicators. For all the data sets under review, a statistically significant improvement in hypervolume is reported by the distance-based implementation of MOABC, along with lower IQR values (in almost all the cases) which suggest less variability in the quality of the inferred Pareto sets. The comparison of median Pareto fronts shows how the distance-based solutions dominate (100%) the solutions obtained by the tree-based implementation on *rbcL_55*, *mtDNA_186*, *ZILLA_500*, achieving also a high set coverage percentage (89.47%) on *RDPII_218*. On the other hand, the tree-based implementation is unable to reach the quality of the solutions inferred by the distance-based MOABC, covering in its best case scenario only a 6.12% of solutions (*RDPII_218*). By examining the shape of the obtained Pareto fronts, it can be observed that the distance-based implementation shows a better distribution of solutions throughout the entire front in terms of number and diversity of solutions. This statement is supported by the results computed by the spacing indicator, which points out the use of distance matrices as a useful tool to obtain a better spread of solutions in all the considered data sets.

Table 6.6 Comparison of tree-building methods for MO-FA under hypervolume

	Hypervolume								
	BIONJ		UPGMA		WPGMA		CLink		Stat.
Dataset	Median	IQR	Median	IQR	Median	IQR	Median	IQR	Sign.?
rbcL_55	**71.4732**	0.0793	67.2740	0.4517	67.7777	0.6646	67.5474	0.8683	✓
mtDNA_186	**70.0040**	0.0076	61.5333	1.5973	61.7376	1.5152	62.3754	1.0082	✓
RDPII_218	**74.7253**	0.0762	66.1608	0.7232	65.9292	1.3663	66.1837	1.2303	✓
ZILLA_500	**72.9587**	0.0228	58.6042	1.5120	60.6829	1.8520	61.7463	1.2510	✓

Table 6.7 Comparison of tree-building methods for MO-FA under set coverage and spacing

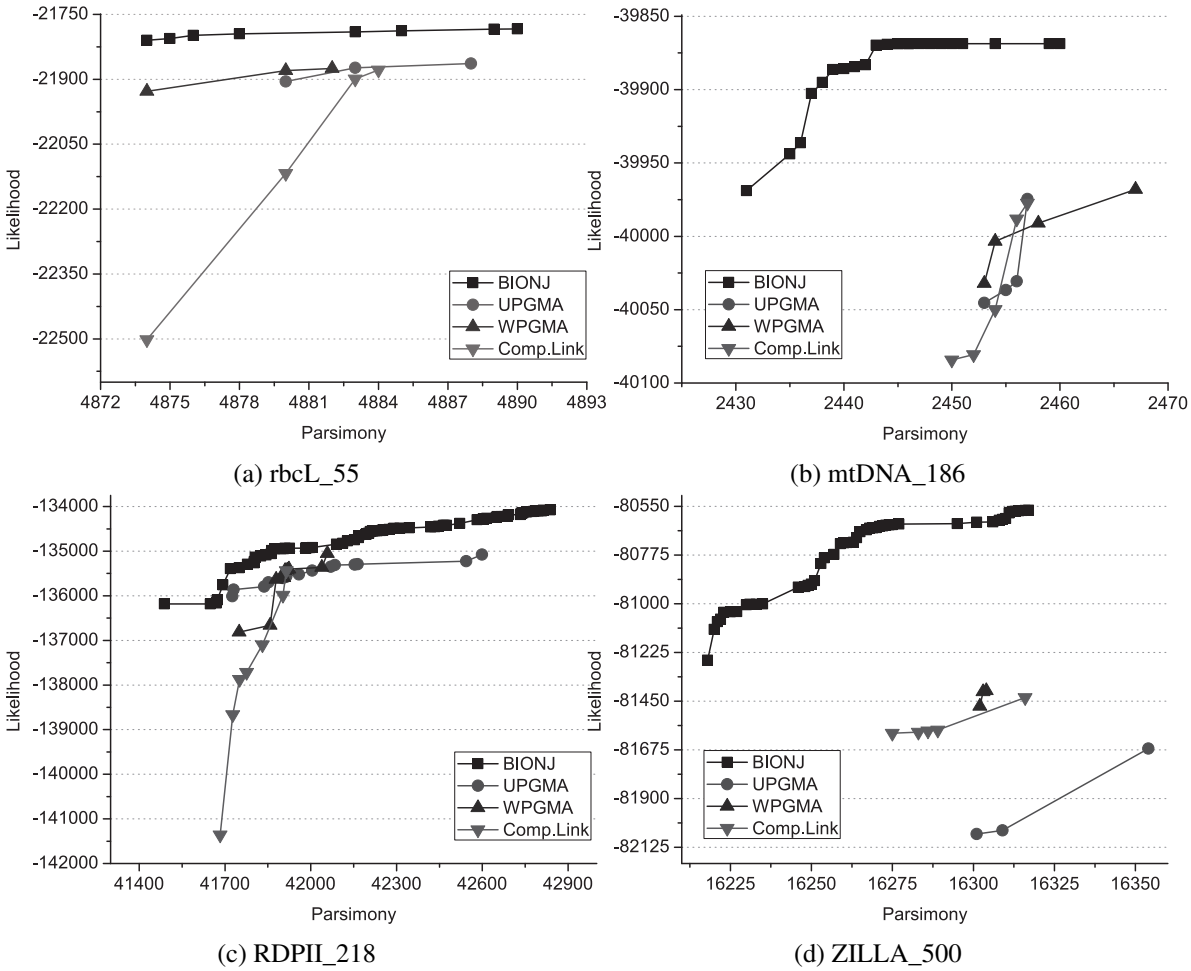
	Coverage Relation				Spacing			
Dataset	SC(BIONJ, UPGMA)	SC(BIONJ, WPGMA)	SC(BIONJ, CLink)	SC(Cluster., BIONJ)	BIONJ	UPGMA	WPGMA	CLink
rbcL_55	**100.0000**	**100.0000**	**100.0000**	0.0000	**0.1284**	0.1437	0.6567	0.4420
mtDNA_186	**100.0000**	**100.0000**	**100.0000**	0.0000	**0.1032**	0.2905	0.2367	0.1968
RDPII_218	**100.0000**	**100.0000**	**100.0000**	0.0000	**0.0325**	0.1183	0.2121	0.1819
ZILLA_500	**100.0000**	**100.0000**	**100.0000**	0.0000	**0.0506**	0.8077	0.4309	0.6209

The obtained results justify the application of the proposed distance-based methodology to undertake phylogenetic searches in a multiobjective context. The applied multiobjective metrics point out how this approach helps to improve the search for high-quality Pareto solutions, along with supporting the diversity of points observed in the front. As a consequence, statistically significant differences in multiobjective performance are reported with regard to traditional tree-based implementations.

6.1.5 Discussing the BIONJ Tree-building Method

As a result of the integration of a distance-based methodology into our metaheuristic search engines, the evaluation of candidate solutions requires to carry out the mapping from the distance matrix space to the phylogenetic tree space. The implementation of this step in our designs has been performed by using the BIONJ tree-building method. However, as remarked in Chapter 2, clustering methods, such as UPGMA, WPGMA, and complete link, may also be applied to this end. In order to justify the choice of BIONJ, we have performed experimentation to verify the quality of the approximation sets generated when using these different tree-building methods. The metaheuristic used to conduct this study was MO-FA, assessing the multiobjective performance attained under BIONJ, UPGMA, WPGMA, and complete link on the data sets *rbcL_55*, *mtDNA_186*, *RDPII_218*, and *ZILLA_500*.

The evaluation of the Pareto sets obtained by using these methods for matrix-tree mapping purposes is shown in Tables 6.6 and 6.7. Table 6.6 details the median hypervolume and IQR values achieved by MO-FA when using BIONJ (columns 2 and 3), UPGMA (columns 4 and 5), WPGMA (columns 6 and 7), and complete link (CLink, columns 8 and 9). Column 10 summarizes the results of applying the statistical testing methodology over the observed hypervolume samples, pointing out if statistically significant differences were attained by BIONJ over the clustering methods. On the other hand, Table 6.7 shows the results of evaluating the median hypervolume Pareto fronts reported



(a) rbcL_55 (b) mtDNA_186

(c) RDPII_218 (d) ZILLA_500

Fig. 6.2 Pareto fronts achieved by MO-FA under different tree-building methods: BIONJ, UPGMA, WPGMA, and complete link clustering

by each tree-building method (represented in Figure 6.2) under set coverage and spacing. Columns 2, 3, and 4 detail the percentage of solutions covered by the BIONJ implementation of MO-FA over the UPGMA, WPGMA, and complete link versions, respectively, while column 5 contains the mean percentage of solutions from BIONJ which are covered by any clustering method. Finally, the results of applying the spacing indicator are provided in columns 6, 7, 8, and 9.

According to the hypervolume values referenced in Table 6.6, a statistically significant improvement over the clustering methods is attained in all the studied data sets when using BIONJ as tree-building method. Moreover, more remarkable differences between BIONJ and the remaining methods are observed for increasing complexities of the input dataset (in terms of number of species). While the selection of the BIONJ tree-building method implies an average improvement of 3.94% in hypervolume when analyzing the *rbcL_55* dataset, this choice has a strong impact for *ZILLA_500*, giving rise to a noticeable mean improvement of 12.61% in hypervolume values over the clustering methods. The significant quality of the Pareto sets inferred when using BIONJ with regard to other

distance methods is also pointed out by the set coverage and spacing metrics. Regarding set coverage, a 100% of the solutions obtained by UPGMA, WPGMA, and complete link are dominated by BIONJ, while no solution in BIONJ fronts is covered by the clustering methods, as shown in Figure 6.2. This figure also makes clear the poor diversity of solutions in the Pareto fronts generated by the UPGMA, WPGMA, and complete link methods. The spacing indicator gives support to this statement, pointing out BIONJ as the best choice to attain satisfying distributions of solutions.

These results reveal the importance of choosing an accurate tree-building method like BIONJ to support the search engines implemented in the metaheuristics, outperforming the clustering methods in all the data sets under study. The poor quality of the solutions attained by the clustering methods is mostly related to the molecular clock assumption. For those biological data sets that do not fit this hypothesis, these methods disturb the topologies codified in the distance matrices, reporting ultrametric trees in which the information of branch length values is lost. On the other hand, a more accurate processing of evolutionary distances is considered in BIONJ, allowing a proper mapping of the distance matrices managed by the metaheuristic to the corresponding phylogenetic trees. In conclusion, BIONJ represents a suitable choice to carry out the tree-building steps in our algorithms.

6.2 Multiobjective Performance Assessment

This section details the comparative assessment of multiobjective metaheuristics when solving the phylogenetic inference problem. To this end, hypervolume, set coverage, and spacing are applied to measure the quality of the Pareto sets inferred by each algorithm on different problem sizes, given by the data sets *rbcL_55*, *mtDNA_186*, *HIV1_192*, *RDPII_218*, *S1482_346*, and *ZILLA_500*. In a first step, comparisons among dominance-based approaches are reported, followed by the evaluation of indicator-based designs. After examining each line separately, comparisons among all the applied metaheuristics are introduced with the aim of identifying what kind of algorithmic design leads to the best overall multiobjective performance for this problem.

6.2.1 Dominance-based Approaches

The first line of algorithmic developments examined in this Thesis is the one which supports the implementation of multiobjective fitness assignment procedures based on Pareto dominance. Up to four dominance-based metaheuristics have been applied to address this problem: MOABC, MO-FA, and the reference MOEAs NSGA-II and SPEA2. The experimental results reported by these algorithms have been evaluated under the three multiobjective metrics considered in our methodology, applying statistical testing to find out if statistically significant differences are observed in the multiobjective performance achieved by the approaches.

Tables 6.8 and 6.9 give account of the results of examining the outcomes of each algorithm under hypervolume, spacing, and set coverage. Firstly, hypervolume and spacing results are shown in Table 6.8. The upper side of the table provides the median hypervolume values and interquartile

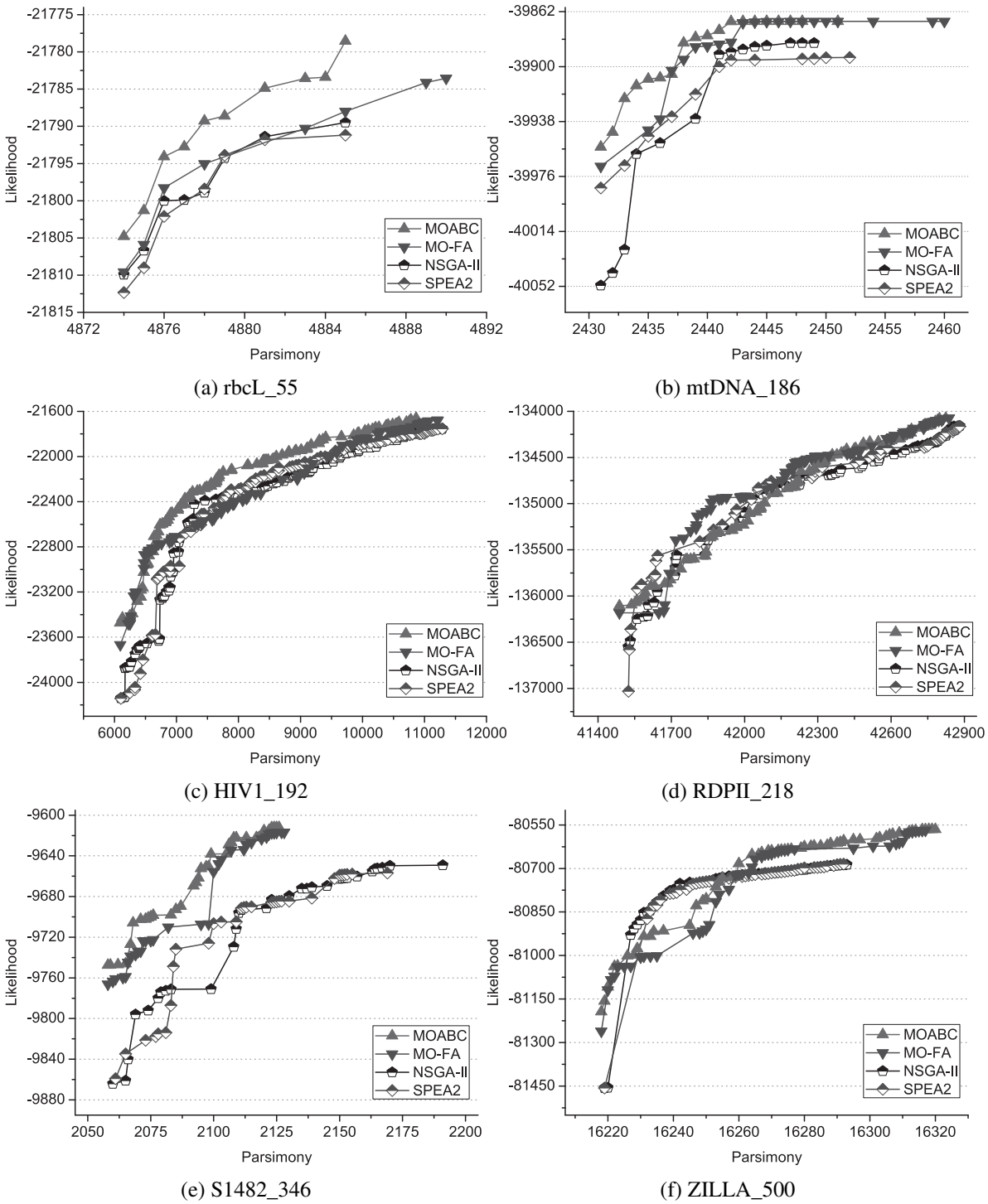
Table 6.8 Comparison of dominance-based approaches under hypervolume and spacing

Hypervolume				
Dataset	MOABC	MO-FA	NSGA-II	SPEA2
rbcL_55	**71.7323±0.0180**	71.4732±0.0793	71.2521±0.1546	71.2270±0.1528
mtDNA_186	**70.0205±0.0140**	70.0040±0.0076	69.7205±0.1120	69.6312±0.1581
HIV1_192	**88.8909±0.2720**	86.6030±0.5368	84.9046±0.4189	85.0813±0.6203
RDPII_218	74.6408±0.0500	**74.7253±0.0762**	73.6252±0.0542	73.7273±0.0662
S1482_346	**80.7980±0.1530**	80.2130±0.3006	75.5470±0.6591	75.3323±0.4869
ZILLA_500	**73.0155±0.0120**	72.9587±0.0228	71.8002±0.0344	71.7999±0.0190
Spacing				
Dataset	MOABC	MO-FA	NSGA-II	SPEA2
rbcL_55	0.0867	0.1284	0.1239	**0.0604**
mtDNA_186	**0.0793**	0.1032	0.1254	0.0995
HIV1_192	**0.0153**	0.0308	0.0206	0.0250
RDPII_218	**0.0168**	0.0325	0.0253	0.0429
S1482_346	**0.0595**	0.0714	0.0601	**0.0595**
ZILLA_500	**0.0309**	0.0506	0.1106	0.1271

Table 6.9 Comparison of dominance-based approaches under set coverage (SC(row, column))

Coverage Relation								
	rbcL_55				*mtDNA_186*			
Algorithm	MOABC	MO-FA	NSGA-II	SPEA2	MOABC	MO-FA	NSGA-II	SPEA2
MOABC		**100.0000**	**100.0000**	**100.0000**		**90.4762**	**100.0000**	**100.0000**
MO-FA	0.0000		75.0000	71.4286	5.2632		92.8571	91.6667
NSGA-II	0.0000	0.0000		71.4286	0.0000	0.0000		58.3333
SPEA2	0.0000	0.0000	25.0000		0.0000	0.0000	35.7143	
	HIV1_192				*RDPII_218*			
Algorithm	MOABC	MO-FA	NSGA-II	SPEA2	MOABC	MO-FA	NSGA-II	SPEA2
MOABC		**85.5556**	**100.0000**	**100.0000**		20.7317	72.2222	64.9123
MO-FA	9.0090		73.2283	54.3103	**65.3061**		**91.6667**	**78.9474**
NSGA-II	0.0000	33.3333		36.2069	25.5102	3.6585		33.3333
SPEA2	0.0000	34.4444	57.4803		41.8367	8.5366	59.7222	
	S1482_346				*ZILLA_500*			
Algorithm	MOABC	MO-FA	NSGA-II	SPEA2	MOABC	MO-FA	NSGA-II	SPEA2
MOABC		**96.8750**	**100.0000**	**100.0000**		**91.6667**	**55.3191**	**61.5385**
MO-FA	0.0000		**100.0000**	**100.0000**	3.4483		**55.3191**	57.6923
NSGA-II	0.0000	0.0000		41.9355	25.8621	27.0833		57.6923
SPEA2	0.0000	0.0000	36.3636		17.2414	20.8333	34.0426	
Mean Results								
Algorithm	MOABC		MO-FA		NSGA-II		SPEA2	
MOABC			**80.8842**		**87.9236**		**87.7418**	
MO-FA	13.8378				81.3452		75.6742	
NSGA-II	8.5621		10.6792				49.8217	
SPEA2	9.8464		10.6357		41.3872			

ranges per dataset observed for MOABC (column 2), MO-FA (column 3), NSGA-II (column 4), and SPEA2 (column 5). The bottom side of the table details the results obtained after applying the spacing indicator over the Pareto fronts obtained in the median hypervolume execution for each



(a) rbcL_55

(b) mtDNA_186

(c) HIV1_192

(d) RDPII_218

(e) S1482_346

(f) ZILLA_500

Fig. 6.3 Comparison of the median Pareto fronts reported by dominance-based approaches: MOABC, MO-FA, NSGA-II, and SPEA2

metaheuristic. Secondly, the pairwise comparison of median Pareto fronts according to the coverage relation is available in Table 6.9. Each entry in this table represents the results of computing SC(X,Y),

Table 6.10 Statistical testing of hypervolume samples for the dominance-based approaches (✓=significant differences, ×=non-significant)

			rbcL_55				*mtDNA_186*	
Algorithm	MOABC	MO-FA	NSGA-II	SPEA2	MOABC	MO-FA	NSGA-II	SPEA2
MOABC		✓	✓	✓		✓	✓	✓
MO-FA			✓	✓			✓	✓
NSGA-II				×				✓
SPEA2								
			HIV1_192				*RDPII_218*	
Algorithm	MOABC	MO-FA	NSGA-II	SPEA2	MOABC	MO-FA	NSGA-II	SPEA2
MOABC		✓	✓	✓		✓	✓	✓
MO-FA			✓	✓			✓	✓
NSGA-II				×				✓
SPEA2								
			S1482_346				*ZILLA_500*	
Algorithm	MOABC	MO-FA	NSGA-II	SPEA2	MOABC	MO-FA	NSGA-II	SPEA2
MOABC		✓	✓	✓		✓	✓	✓
MO-FA			✓	✓			✓	✓
NSGA-II				×				×
SPEA2								

where X is the algorithm specified in the row and Y the algorithm in the column. For example, the first row indicates the percentage of solutions from MO-FA (columns 3 and 7), NSGA-II (columns 4 and 8), and SPEA2 (columns 5 and 9) which are covered by the solutions obtained by MOABC. The representation of the median Pareto fronts reported by these algorithms is given by Figure 6.3.

The hypervolume comparison points out the relevance of applying swarm intelligence algorithms to carry out real phylogenetic analyses. More specifically, MOABC achieves the best hypervolume scores for the instances *rbcL_55* (covering a 71.73% of the objective space), *mtDNA_186* (70.02%), *HIV1_192* (88.89%), *S1482_346* (80.80%), and *ZILLA_500* (73.02%), representing the leading algorithm among the studied dominance-based approaches. For *RDPII_218*, MO-FA attains the most satisfying performance from a hypervolume perspective (74.73%), while obtaining the second best values in the remaining instances. Regarding the reference MOEAs, although significant hypervolume values are retrieved by NSGA-II and SPEA2 (over 69.72% and 69.63%, respectively, in all the data sets), the solutions attained by these algorithms do not reach the overall quality of the ones reported by the swarm intelligence designs. In fact, increased differences in hypervolume between the swarm intelligence designs and the evolutionary algorithms are observed in the instances with the most species (*S1482_346*, *ZILLA_500*), as well as in those instances which involve a high number of solutions in the Pareto front (*HIV1_192* and *RDPII_218*).

In this context, a key question to be addressed consists of determining if the differences observed in hypervolume are statistically significant or not. For this purpose, Table 6.10 shows the results of applying our statistical testing methodology to make pairwise comparisons of hypervolume samples between the assessed dominance-based approaches. Focusing on MOABC, this methodology

reports statistically significant improvements over MO-FA in all the data sets where this metaheuristic achieves the best hypervolume values. The difference between MO-FA and MOABC on *RDPII_218* is also statistically significant, representing therefore the only scenario where MOABC do not achieve the best multiobjective performance. The comparison of MOABC with NSGA-II and SPEA2 gives account of a statistically significant improvement over the reference MOEAs in all the scenarios under study. This statistical testing also points out MO-FA as the second leading algorithm, reporting statistically significant differences with regard to NSGA-II and SPEA2. Non-significant differences from a statistical perspective are only found in the comparison between NSGA-II and SPEA2, particularly, for the instances *rbcL_55*, *HIV1_192*, *S1482_346*, and *ZILLA_500*.

According to the obtained results, these swarm intelligence designs go a step further over traditional evolutionary approaches in the search for high-quality Pareto solutions to the tackled problem. The improved search capabilities can be justified by analyzing the algorithmic designs of these approaches. Searches in MOABC are based on the combination of exploration and exploitation steps implemented by means of scout and employed/onlooker bees. While promising solutions are exploited by performing employed and onlooker searches, the scout bee step allows the algorithm to conduct the analysis over undiscovered regions of the search space by reinitializing the search when local optima are detected. The sharing of information between bees also represents an improved feature over traditional evolutionary designs, as the search engine addresses the problem by considering all the information gathered by the entire swarm.

As for MO-FA, this algorithm applies the concept of swarm intelligence to reconstruct new solutions by transmitting the knowledge gathered by all the fireflies in the swarm during the attraction step. In this way, fireflies interact with each other to conduct the search for optimal phylogenies, introducing a balance between exploitation and exploration of the search space by using the concepts of attractiveness, light intensity and randomization in the firefly movement. This design gives as a result an efficient sharing of information among fireflies, which allows MO-FA to improve the evolutionary designs. However, a robust mechanism to address local optima as the one implemented in the MOABC scout bee step cannot be found in the algorithmic design of MO-FA. This mechanism represents one of the key features that differentiate these algorithms, along with the probabilistic exploitation of the best found solutions performed in the onlooker bee step.

After evaluating hypervolume results, the study of spacing values for each metaheuristic is now undertaken. By analyzing Table 6.8 and the Pareto fronts represented in Figure 6.3, it can be observed how MOABC attains the most satisfying distribution of solutions in the Pareto fronts generated for the instances *mtDNA_186*, *HIV1_192*, *RDPII_218*, *S1482_346*, and *ZILLA_500*, while only being improved by SPEA2 on *rbcL_55*. SPEA2 also reports good distributions in terms of spacing for *mtDNA_186* and *S1482_346*. For the instances which involve the most number of solutions in the fronts (*HIV1_192* and *RDPII_218*), the second best spacing scores are obtained by NSGA-II. On the other hand, MO-FA only achieves a good spacing result on *ZILLA_500*.

From a diversity perspective, the spacing indicator points out MOABC as the metaheuristic which achieved the most satisfying spread of Pareto solutions in overall terms. In this case, MO-FA is not

well-supported as an accurate metaheuristic to attain uniformly distributed Pareto fronts from the spacing perspective. This issue is motivated by the lack of density estimation information in the firefly attraction step, which is purely governed by the comparison of solutions under dominance. Although high-quality solutions are obtained, as pointed out by the hypervolume indicator, less diversity in the inferred non-dominated solutions is observed with regard to MOABC, NSGA-II, and SPEA2. Regarding the reference MOEAs, it is remarkable how the density mechanism implemented in SPEA2 deals better with data sets whose Pareto fronts involve a low number of solutions (*rbcL_55* and *mtDNA_186*), while the crowding distance measurement in NSGA-II attains better spacing results for *HIV1_192* and *RDPII_218*, the ones which show higher numbers of points.

Finally, the coverage relation, as shown in Table 6.9, also gives account of the significant multiobjective performance achieved by MOABC, which represents the leading dominance-based approach under the three considered multiobjective metrics. Considering the results reported by this metric in all the data sets, MOABC covers average percentages of 80.88%, 87.92%, and 87.74% of the solutions inferred by MO-FA, NSGA-II, and SPEA2, respectively. MOABC is only dominated from a set coverage point of view on *RDPII_218*, as a percentage of 65.31% of the solutions obtained by MOABC are covered by the metaheuristic which achieves the best solutions in this dataset, MO-FA. The results of this binary quality indicator for MO-FA points out that this swarm intelligence algorithm is also able to outperform the reference MOEAs, covering average percentages of 81.35% and 75.67% of the solutions inferred by NSGA-II and SPEA2. The pairwise comparison of the two evolutionary designs suggests improved overall performance achieved by NSGA-II over SPEA2, obtaining an average set coverage score of 49.82% (in comparison with the 41.39% of solutions from the fronts inferred by NSGA-II which are covered by SPEA2).

In conclusion, the study of dominance-based approaches point outs the improved multiobjective performance attained by our swarm intelligence designs over the widely-used reference MOEAs NSGA-II and SPEA2 when undertaking multiobjective phylogenetic searches. The three performance metrics applied to measure multiobjective quality agree on suggesting MOABC as the best dominance-based metaheuristic design, achieving statistically significant improvements over MO-FA, NSGA-II, and SPEA2 in most of the scenarios under evaluation.

6.2.2 Indicator-based Approaches

The integration of quality indicators into fitness assignment procedures represents the key feature behind the second line of multiobjective algorithmic developments examined in this Thesis. Now, we undertake the comparison of multiobjective performance between the swarm intelligence design of IMOBA and the indicator-based reference MOEA known as IBEA. As in the dominance-based case, the results obtained by these algorithms have been assessed under the hypervolume, set coverage, and spacing metrics, conducting statistically reliable comparisons to decide which indicator-based approach represents a better choice to address the problem. Tables 6.11 and 6.12 report the assessment of multiobjective results under the three considered indicators.

Table 6.11 Comparison of indicator-based approaches under hypervolume and spacing

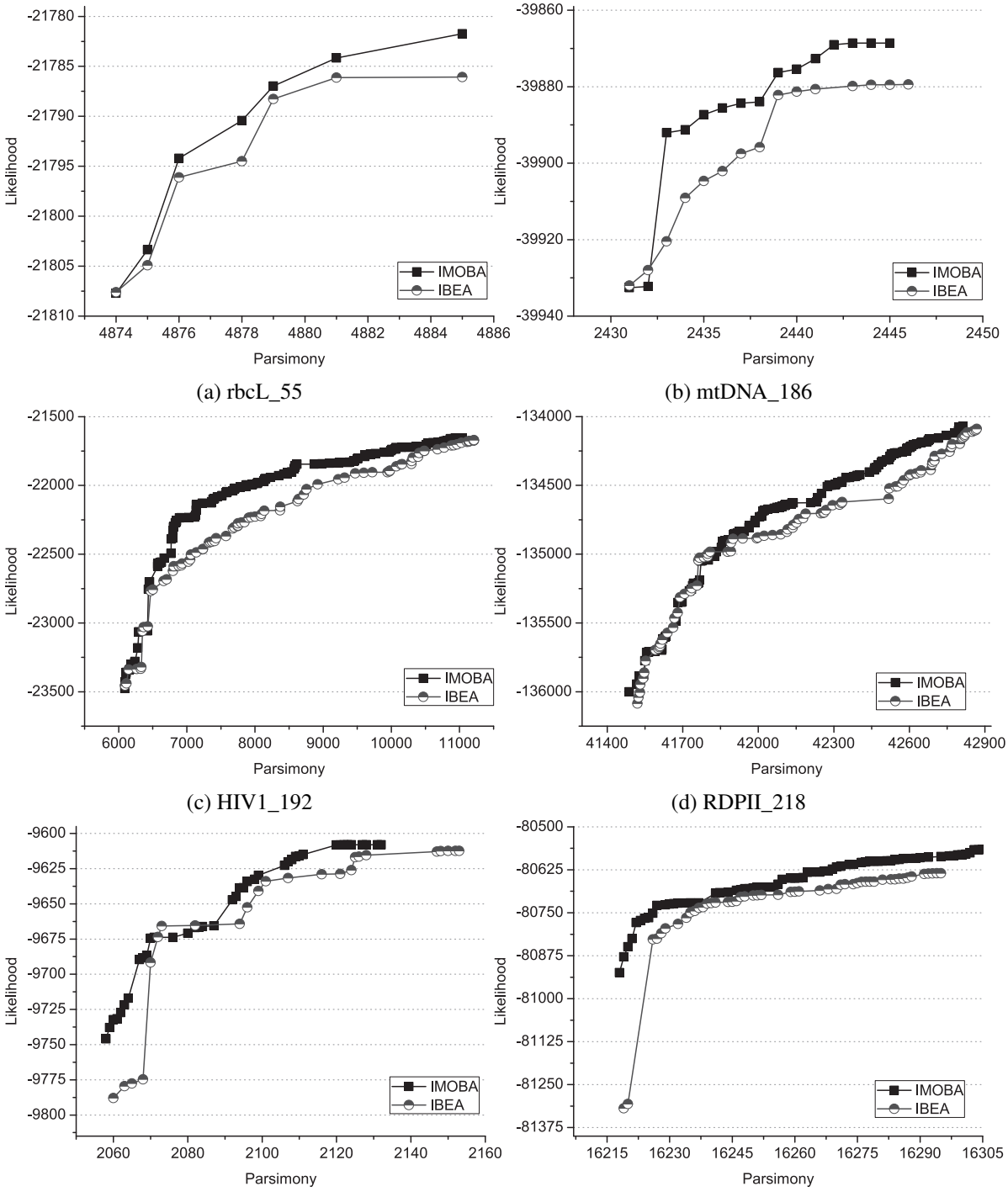
	Hypervolume		Spacing	
Dataset	IMOBA	IBEA	IMOBA	IBEA
rbcL_55	**71.6034±0.0178**	71.4319±0.0519	**0.0884**	0.0942
mtDNA_186	**70.0339±0.0054**	69.8303±0.0844	0.0958	**0.0777**
HIV1_192	**90.5596±0.1872**	87.9924±0.4461	**0.0191**	0.0263
RDPII_218	**75.0646±0.0375**	74.3017±0.0617	**0.0159**	0.0206
S1482_346	**81.3547±0.1815**	80.0848±0.3214	**0.0529**	0.0992
ZILLA_500	**73.0815±0.0046**	72.3369±0.0423	**0.0277**	0.1075

Table 6.12 Comparison of indicator-based approaches under set coverage (SC(row, column))

Coverage Relation						
	rbcL_55		*mtDNA_186*		*HIV1_192*	
Algorithm	IMOBA	IBEA	IMOBA	IBEA	IMOBA	IBEA
IMOBA		**85.7143**		**86.6700**		**93.9024**
IBEA	14.2857		13.3333		1.4184	
	RDPII_218		*S1482_346*		*ZILLA_500*	
Algorithm	IMOBA	IBEA	IMOBA	IBEA	IMOBA	IBEA
IMOBA		**75.2941**		**87.5000**		**100.0000**
IBEA	8.8000		16.6667		0.0000	
	Mean Results					
Algorithm	IMOBA			IBEA		
IMOBA				**88.1801**		
IBEA	9.0840					

The comparison between IMOBA and IBEA under hypervolume and spacing is given in Table 6.11. In this table, median hypervolume values and interquartile ranges are given by columns 2 and 3, while columns 4 and 5 detail the results of applying the spacing indicator over the median hypervolume Pareto fronts reported by IMOBA and IBEA, respectively. Regarding the evaluation under the coverage relation, Table 6.12 provides set coverage scores for each dataset, where each entry represents the percentage of solutions from the metaheuristic specified in the column which is covered by the solutions of the algorithm indicated in the row. The median Pareto fronts observed in our experimentation with IMOBA and IBEA are graphically represented in Figure 6.4.

According to the scores reported by the hypervolume indicator, IMOBA achieves a noticeable improvement over IBEA in all the considered scenarios. Under this metric, the Pareto solutions obtained by the swarm intelligence approach dominate percentages above 70.03% of the objective area bounded by the reference point. The performance obtained on *HIV1_192* is especially remarkable, as IMOBA succeeds in dominating a 90.56% of the considered objective region for this dataset. IBEA also gets meaningful performance in the different instances under study, although the reported hypervolume scores are lower than the ones achieved by IMOBA. In fact, the statistical testing of hypervolume samples (as given in Table 6.13) gives account of statistically significant differences



(a) rbcL_55 (b) mtDNA_186

(c) HIV1_192 (d) RDPII_218

(e) S1482_346 (f) ZILLA_500

Fig. 6.4 Comparison of the median Pareto fronts reported by indicator-based approaches: IMOBA and IBEA

between IMOBA and IBEA for all the data sets. Furthermore, the representation of Pareto fronts in Figure 6.4 shows the significant convergence and diversity features observed in the solutions inferred

Table 6.13 Statistical testing of hypervolume samples for the indicator-based approaches (✓=significant differences, ×=non-significant)

	rbcL_55		*mtDNA_186*		*HIV1_192*	
Algorithm	IMOBA	IBEA	IMOBA	IBEA	IMOBA	IBEA
IMOBA		✓		✓		✓
IBEA						
	RDPII_218		*S1482_346*		*ZILLA_500*	
Algorithm	IMOBA	IBEA	IMOBA	IBEA	IMOBA	IBEA
IMOBA		✓		✓		✓
IBEA						

by IMOBA. The relevant spread of points in the front achieved by this metaheuristic is confirmed by the spacing indicator, which suggests improved scores for the fronts generated by IMOBA on *rbcL_55*, *HIV1_192*, *RDPII_218*, *S1482_346*, and *ZILLA_500*. On the other hand, IBEA obtains the best spacing value on *mtDNA_186*. Finally, the comparison of Pareto fronts under the coverage relation confirms the position of IMOBA as the most satisfying indicator-based approach to generate high-quality multiobjective phylogenies. For all the analyzed instances, the Pareto fronts from IMOBA dominate percentages over 75.29% of the solutions generated by IBEA. In its best case scenario, the reference indicator-based MOEA covers at most a 16.67% of the solutions inferred by IMOBA. In average, IBEA is able to cover a 9.08% of IMOBA solutions, while the swarm intelligence approach obtains a significant set coverage score of 88.18% with regard to IBEA.

In summary, hypervolume, spacing, and set coverage agree on pointing out IMOBA as the most successful indicator-based approach for dealing with this complex bioinformatics problem. The swarm intelligence principles which define the algorithmic design of this metaheuristic allow IMOBA to tackle the reconstruction of phylogenetic hypotheses by defining different mechanisms (movement, exploitation, and exploration), which take into account the knowledge gathered by the entire bat population to improve search capabilities. On the one hand, the movement step is defined with the aim of generating new bats which combine the information stored in the most promising *popSize* individuals, contending with the original ones by means of the considered quality indicator. On the other hand, the pulse emission rates introduce different priorities to the exploration and exploitation steps throughout the execution of the algorithm, evolving the way the proposal carries out the processing of the search space. In this sense, the exploitation mechanism considers the current average loudness observed in the population to generate new solutions from promising ones, while the exploration procedure deals with local optima issues by introducing solutions obtained from new starter topologies. In this way, the algorithm is able to reach a balance between the exploitation of the most satisfying solutions and the exploration of further promising regions of the search space, in accordance with the hunting strategies shown by bats in nature.

Therefore, the analysis of the multiobjective performance attained by the indicator-based approaches suggests how the swarm intelligence mechanisms in IMOBA lead to an improved process-

ing of the search space and, as a consequence, improved solution quality with regard to the evolutionary techniques in IBEA. In this sense, IMOBA succeeds in reporting very significant hypervolume results in all the data sets under analysis, while obtaining satisfying distributions of solutions and high percentages of set coverage in comparison to the reference indicator-based MOEA.

6.2.3 Dominance vs. Indicators

After assessing each metaheuristic line separately, a direct comparison between indicator-based and dominance-based approaches is now conducted. The main goal behind this comparative study is to decide, among the six metaheuristics applied to phylogenetic inference in this research, which approach is the one that leads to the best overall performance from a multiobjective point of view. Table 6.14 encloses the overall scores attained by the metaheuristics under hypervolume and spacing, while Table 6.15 provides the statistical comparison of hypervolume samples for each pair of metaheuristics. Finally, Table 6.16 reports the results of comparing the six multiobjective algorithms each other according to the coverage relation indicator.

Firstly, we focus on evaluating hypervolume performance. By classifying the metaheuristics according to the class of bioinspired technique integrated in their search engines (swarm intelligence and evolutionary designs), we can verify the benefits of applying indicator-based designs to phylogenetic inference. In general terms, these approaches outperform the dominance-based algorithms (IMOBA over MOABC and MO-FA, and IBEA over NSGA-II and SPEA2) from a hypervolume perspective in almost all the considered scenarios. In this sense, hypervolume points out IMOBA as the most accurate multiobjective design in overall terms, obtaining the best scores on *mtDNA_186*, *HIV1_192*, *RDPII_218*, *S1482_346*, and *ZILLA_500*. For the simplest dataset (*rbcL_55*), this algorithm attains the second best score (71.60), very close to the performance reported by MOABC (71.73). IBEA also attained noticeable results in the comparison, outperforming the dominance-based reference MOEAs (NSGA-II and SPEA2) in all the data sets. Furthermore, the results reported on *HIV1_192* suggest that IBEA is even able to improve the results reported by a swarm intelligence algorithm, MO-FA, representing a meaningful observation due to the fact that this dataset is the one that involves the highest number of points in the Pareto front.

According to our statistical testing methodology, statistically significant differences between IMOBA and MOABC/MO-FA and IBEA and NSGA-II/SPEA are found in all the considered data sets. Moreover, significant differences are reported for the leading algorithm (IMOBA) with regard to the remaining metaheuristics, suggesting the relevance of the results attained by the bat-inspired design. The comparison between IBEA and MO-FA gives account of non-significant differences for the *rbcL_55* dataset, which confirms the noticeable performance of this indicator-based MOEA not only with regard to the widely-used NSGA-II and SPEA2, but also in comparison with a more advanced bioinspired search engine like the one implemented in MO-FA.

Regarding spacing, this indicator also points out the relevance of the indicator-based designs, which gives rise to the most satisfying distribution of Pareto solutions in four of the six instances un-

Table 6.14 Overall multiobjective comparison under hypervolume and spacing

Hypervolume						
Dataset	IMOBA	IBEA	MOABC	MO-FA	NSGA-II	SPEA2
rbcL_55	71.603±0.018	71.432±0.052	**71.732±0.018**	71.473±0.079	71.252±0.155	71.227±0.153
mtDNA_186	**70.034±0.005**	69.830±0.084	70.021±0.014	70.004±0.008	69.721±0.112	69.631±0.158
HIV1_192	**90.560±0.187**	87.992±0.446	88.891±0.272	86.603±0.537	84.905±0.419	85.081±0.620
RDPII_218	**75.065±0.038**	74.302±0.062	74.641±0.050	74.725±0.076	73.625±0.054	73.727±0.066
S1482_346	**81.355±0.182**	80.085±0.321	80.798±0.153	80.213±0.301	75.547±0.659	75.332±0.487
ZILLA_500	**73.082±0.005**	72.337±0.042	73.016±0.012	72.959±0.023	71.800±0.034	71.800±0.019
Spacing						
Dataset	IMOBA	IBEA	MOABC	MO-FA	NSGA-II	SPEA2
rbcL_55	0.088	0.094	0.087	0.128	0.124	**0.060**
mtDNA_186	0.096	**0.078**	0.079	0.103	0.125	0.100
HIV1_192	0.019	0.026	**0.015**	0.031	0.021	0.025
RDPII_218	**0.016**	0.021	0.017	0.033	0.025	0.043
S1482_346	**0.053**	0.099	0.060	0.071	0.060	0.060
ZILLA_500	**0.028**	0.108	0.031	0.051	0.111	0.127

Table 6.15 Overall statistical testing of hypervolume samples (✓=significant differences, ×=non-significant)

rbcL_55							*mtDNA_186*					
Algorithm	IMOBA	IBEA	MOABC	MO-FA	NSGA-II	SPEA2	IMOBA	IBEA	MOABC	MO-FA	NSGA-II	SPEA2
IMOBA		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓
IBEA			✓	×	✓	✓			✓	✓	✓	✓
MOABC				✓	✓	✓				✓	✓	✓
MO-FA					✓	✓					✓	✓
NSGA-II						×						✓
SPEA2												
HIV1_192							*RDPII_218*					
Algorithm	IMOBA	IBEA	MOABC	MO-FA	NSGA-II	SPEA2	IMOBA	IBEA	MOABC	MO-FA	NSGA-II	SPEA2
IMOBA		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓
IBEA			✓	✓	✓	✓			✓	✓	✓	✓
MOABC				✓	✓	✓				✓	✓	✓
MO-FA					✓	✓					✓	✓
NSGA-II						×						✓
SPEA2												
S1482_346							*ZILLA_500*					
Algorithm	IMOBA	IBEA	MOABC	MO-FA	NSGA-II	SPEA2	IMOBA	IBEA	MOABC	MO-FA	NSGA-II	SPEA2
IMOBA		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓
IBEA			✓	✓	✓	✓			✓	✓	✓	✓
MOABC				✓	✓	✓				✓	✓	✓
MO-FA					✓	✓					✓	✓
NSGA-II						×						×
SPEA2												

der study: *mtDNA_186* (IBEA), *RDPII_218*, *S1482_346*, and *ZILLA_500* (IMOBA). The dominance-based approaches provide the best spacing values on *rbcL_55* (SPEA2) and *HIV1_192* (MOABC), achieving this last metaheuristic noticeable performance in all the data sets (reporting the second best spacing values in five instances). Finally, the pairwise comparison of fronts under the coverage relation indicator highlights the significant quality of the solutions reported by IMOBA. With regard to the dominance-based approaches, IMOBA succeeds in obtaining average set coverage percentages of 81.99% and 95.12% over MOABC and MO-FA, while clearly dominating NSGA-II and SPEA2

Table 6.16 Overall multiobjective comparison under set coverage

rbcL_55						
Algorithm	IMOBA	IBEA	MOABC	MO-FA	NSGA-II	SPEA2
IMOBA		**85.714**	20.000	**100.000**	**100.000**	**100.000**
IBEA	14.286		10.000	75.000	**100.000**	**100.000**
MOABC	**71.429**	**85.714**		**100.000**	**100.000**	**100.000**
MO-FA	0.000	0.000	0.000		75.000	71.429
NSGA-II	0.000	0.000	0.000	0.000		71.429
SPEA2	0.000	0.000	0.000	0.000	25.000	
mtDNA_186						
Algorithm	IMOBA	IBEA	MOABC	MO-FA	NSGA-II	SPEA2
IMOBA		**86.670**	**89.474**	**100.000**	**100.000**	**100.000**
IBEA	13.333		36.842	38.095	**100.000**	**100.000**
MOABC	13.333	53.333		90.476	**100.000**	**100.000**
MO-FA	0.000	33.333	5.263		92.857	91.667
NSGA-II	0.000	0.000	0.000	0.000		58.333
SPEA2	0.000	0.000	0.000	0.000	35.714	
HIV1_192						
Algorithm	IMOBA	IBEA	MOABC	MO-FA	NSGA-II	SPEA2
IMOBA		**93.902**	**95.496**	**100.000**	**100.000**	**100.000**
IBEA	1.418		17.117	82.222	99.213	94.828
MOABC	1.418	82.927		85.556	**100.000**	**100.000**
MO-FA	0.000	17.073	9.009		73.228	54.310
NSGA-II	0.000	0.000	0.000	33.333		36.207
SPEA2	0.000	3.659	0.000	34.444	57.480	
RDPII_218						
Algorithm	IMOBA	IBEA	MOABC	MO-FA	NSGA-II	SPEA2
IMOBA		**75.294**	**93.878**	**70.732**	**100.000**	**98.246**
IBEA	8.800		45.918	28.049	94.444	68.421
MOABC	1.600	45.882		20.732	72.222	64.912
MO-FA	8.000	57.647	65.306		91.667	78.947
NSGA-II	0.000	1.176	25.510	3.659		33.333
SPEA2	0.000	24.706	41.837	8.537	59.722	
S1482_346						
Algorithm	IMOBA	IBEA	MOABC	MO-FA	NSGA-II	SPEA2
IMOBA		**87.500**	**100.000**	**100.000**	**100.000**	**100.000**
IBEA	16.667		30.556	53.125	**100.000**	**100.000**
MOABC	0.000	75.000		96.875	**100.000**	**100.000**
MO-FA	0.000	29.167	0.000		**100.000**	**100.000**
NSGA-II	0.000	0.000	0.000	0.000		41.936
SPEA2	0.000	0.000	0.000	0.000	36.364	
ZILLA_500						
Algorithm	IMOBA	IBEA	MOABC	MO-FA	NSGA-II	SPEA2
IMOBA		**100.000**	**93.103**	**100.000**	**100.000**	**100.000**
IBEA	0.000		27.586	37.500	**100.000**	**100.000**
MOABC	0.000	54.000		91.667	55.319	61.539
MO-FA	0.000	50.000	3.448		55.319	57.692
NSGA-II	0.000	0.000	25.862	27.083		57.692
SPEA2	0.000	0.000	17.241	20.833	34.043	
Mean Results						
Algorithm	IMOBA	IBEA	MOABC	MO-FA	NSGA-II	SPEA2
IMOBA		**88.180**	**81.992**	**95.122**	**100.000**	**99.708**
IBEA	9.084		28.003	52.332	98.943	93.875
MOABC	14.630	66.143		80.884	87.924	87.742
MO-FA	1.333	31.203	13.838		81.345	75.674
NSGA-II	0.000	0.196	8.562	10.679		49.822
SPEA2	0.000	4.727	9.846	10.636	41.387	

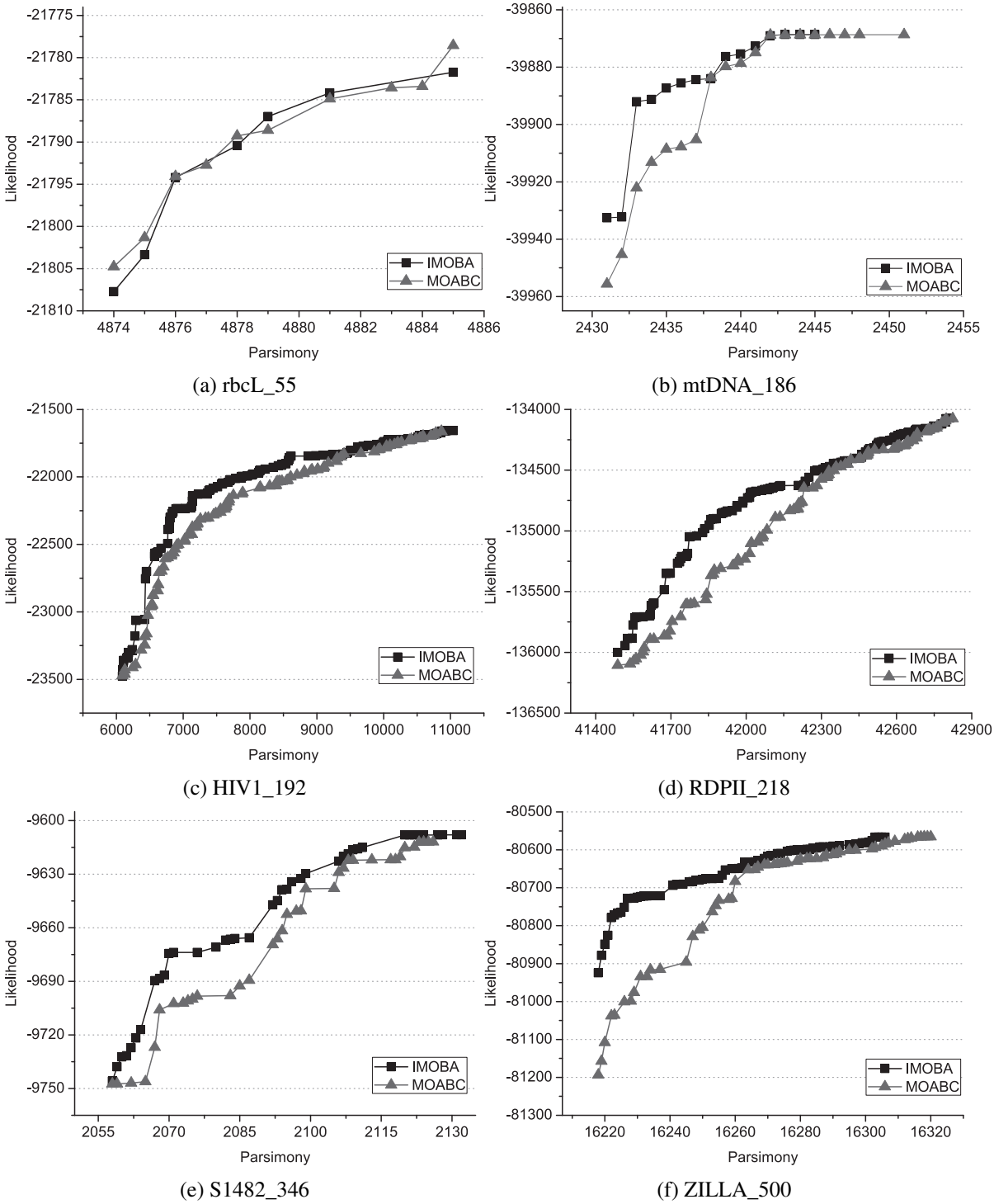
attending to this indicator (100% and 99.71%). Regarding IBEA, although it is unable to deal with the significant solution quality of MOABC (28.00%), the indicator-based MOEA dominates average percentages of 98.94% and 93.88% of the solutions in NSGA-II and SPEA2, while achieving a set coverage of 52.33% over MO-FA. On the other hand, the dominance-based approaches show lower set coverage percentages, representing MOABC the most satisfying approach by covering average solution percentages of 14.63% and 66.143% with regard to IMOBA and IBEA.

Considering the overall scores returned by the three performance metrics, we can identify a leading metaheuristic for each development line of multiobjective algorithms: IMOBA in the indicator-based side and MOABC in the dominance-based side. When comparing these two metaheuristics directly, both hypervolume and set coverage give account of the significant performance achieved by IMOBA over MOABC in almost all the data sets, while similar results are reported attending to the spacing metric (IMOBA achieving better spread of solutions in the data sets with higher number of species and MOABC improving the ones with lower number of species). In order to understand the conclusions hinted by these quality indicators, Figure 6.5 represents the graphical comparison of the median Pareto fronts achieved by IMOBA and MOABC. For the data sets where IMOBA reports the best multiobjective results, it can be observed that both metaheuristics verify comparable results in the region of the Pareto fronts most supported by the likelihood criterion (second half of the front). On the other hand, IMOBA tends to dominate MOABC in the parsimony region (first half of the front) by improving the likelihood scores of the resulting solutions (which is especially noticeable on *ZILLA_500*). This fact denotes the success of the indicator on guiding the robust bat-inspired search engine towards solutions that maximize the quality of the Pareto approximation set.

Concluding Remarks

The evaluation of multiobjective performance has pointed out that the combination of quality indicators and swarm intelligence in IMOBA leads to an algorithmic design which tackles the reconstruction of multiobjective phylogenetic hypotheses in a successful way. In overall, IMOBA attains the best Pareto sets, outperforming IBEA, MOABC, MO-FA, NSGA-II, and SPEA2 from a multiobjective point of view. From this comparative study, two significant facts from a bioinspired computing perspective are observed. Firstly, under swarm intelligence principles, bioinspired search engines can benefit from the knowledge gathered by all the individuals in the swarm to address high-complexity optimization problems, like the one addressed in this Thesis. This statement is supported by the comparisons of multiobjective performance between traditional evolutionary designs (IBEA, NSGA-II, SPEA2) and swarm intelligence approaches (IMOBA, MOABC, MO-FA). On the dominance-based side, our experiments suggest that MOABC and MO-FA give rise to improved hypervolume values over NSGA-II and SPEA2. Regarding the indicator-based designs, swarm intelligence also gives as a result improved solution quality, as shown by the comparison between IMOBA and IBEA.

Secondly, the introduction of quality indicators to guide the search represents a significant improvement over traditional Pareto rankings and density estimation procedures for this problem. This



(a) rbcL_55 (b) mtDNA_186

(c) HIV1_192 (d) RDPII_218

(e) S1482_346 (f) ZILLA_500

Fig. 6.5 Comparison of the median Pareto fronts reported by the best dominance-based and indicator-based metaheuristics

conclusion can be applied not only to traditional evolutionary algorithms (IBEA vs. NSGA-II and SPEA2), but also to swarm intelligence designs. As shown in the comparisons, the indicator-based

design of IMOBA achieves the best overall results, improving the dominance-based swarm intelligence designs (MOABC and MO-FA) in almost all the considered scenarios.

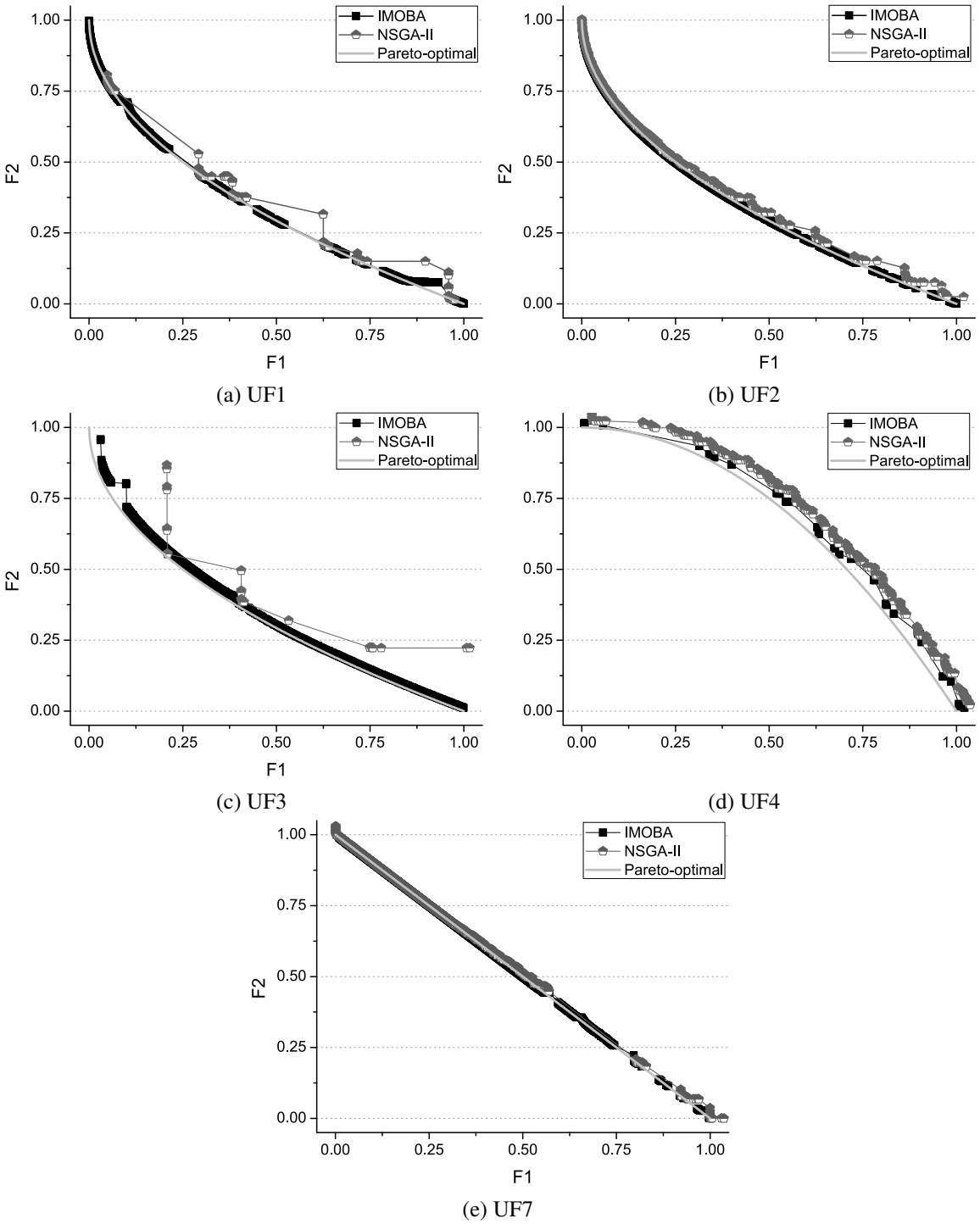
Phylogenetic inference represents an optimization problem whose solution must deal with very large search spaces and accuracy issues to provide relevant results [170]. As a result, this problem is said to be difficult from a bioinspired computing point of view [40]. This kind of challenging optimization problems requires advanced search strategies to attain high-quality solutions. As shown in our comparative study, the algorithmic design of IMOBA leads to significant results attending to multiobjective performance, suggesting that the combination of indicator-based searches and swarm intelligence provides suitable mechanisms to address such problems.

6.2.4 Assessing IMOBA Performance on Test Problems

In order to substantiate the argument on the significance of IMOBA, we have studied the performance attained by this method in different multiobjective benchmarks. For this purpose, we have considered several problem instances from the *2009 IEEE Congress on Evolutionary Computation* (CEC09) algorithm contest [193]. The problems proposed in this contest aim to resemble complicated real-life problems with the idea of verifying the robustness of multiobjective search engines. We have chosen five test instances (UF1, UF2, UF3, UF4, and UF7) which represent MOPs of different nature, attending to the characteristics of their search spaces and the shapes of their Pareto-optimal fronts (concave, convex, and linear). According to the mathematical formulation of these problems (available in [193]), the number of decision variables is 30 and the objectives are to be minimized.

We have applied IMOBA, along with the reference MOEAs IBEA, NSGA-II, and SPEA2, to solve these problems, considering the following implementation details. For IMOBA, we used the algorithmic scheme detailed in the previous chapter, adapting the movement, exploitation, and exploration steps to work with arrays of 30 decision variables (individual representation). The implementations of IBEA, NSGA-II, and SPEA2 were developed in accordance with the guidelines from [161], introducing a SBX crossover and a polynomial mutation procedure as evolutionary operators. These algorithms were configured by making parametric studies on the considered test instances. The input settings for IMOBA involve a population size of 100 bats, a mutation probability of 25%, a frequency range $[f_{min}, f_{max}]$ of [0.5, 2], an initial loudness A_0 of 0.10, a loudness update factor α of 0.9, a maximum pulse emission rate r_{max} of 0.25, and a pulse update factor of 0.3. The settings for IBEA, NSGA-II, and SPEA2 define a population size of 100 individuals, a crossover probability of 90% (with a SBX distribution index = 5), and a mutation probability of 5% (with a polynomial distribution index = 15). In addition, the archive size in SPEA2 was set to 100 individuals, while the scaling factor κ and the I_{HD} reference point Z_{ref} in IMOBA and IBEA were set to 0.05 and (2, 2). Following the contest rules, the stop criterion was set to 300,000 evaluations for each test problem.

Under these assumptions, our experiments involved 31 independent runs of each algorithm per test problem. The measurement of multiobjective performance was carried out by using hypervolume, calculating this quality indicator with regard to the reference point (1.1, 1.1). Table 6.17 details



(a) UF1

(b) UF2

(c) UF3

(d) UF4

(e) UF7

Fig. 6.6 Median Pareto fronts reported by IMOBA and NSGA-II for the CEC09 test problems

the median hypervolume values scored by IMOBA, IBEA, NSGA-II, and SPEA2 on the test problems under study. In addition, Table 6.18 shows the results of applying the pairwise statistical testing of hypervolume samples, in order to find out if statistically significant differences in multiobjective

Table 6.17 Test problems - comparisons of hypervolume performance

Problem	I_H(IMOBA)	I_H(IBEA)	I_H(NSGA-II)	I_H(SPEA2)
UF1	**85.30±1.95**	80.44±3.49	77.41±5.83	76.18±5.16
UF2	**87.33±0.19**	85.16±0.57	84.14±0.38	80.16±0.40
UF3	**84.95±1.62**	68.04±5.80	67.56±5.23	67.73±3.10
UF4	**47.60±0.38**	46.76±0.58	46.79±0.14	44.10±1.06
UF7	**69.46±1.63**	67.26±1.82	65.44±2.48	62.68±2.06

Table 6.18 Test problems - statistical testing of hypervolume samples (√=significant differences, ×=non-significant)

	UF1				UF2			
Method	IMOBA	IBEA	NSGA-II	SPEA2	IMOBA	IBEA	NSGA-II	SPEA2
IMOBA		√	√	√		√	√	√
IBEA			√	√			√	√
NSGA-II				×				√
SPEA2								
	UF3				UF4			
Method	IMOBA	IBEA	NSGA-II	SPEA2	IMOBA	IBEA	NSGA-II	SPEA2
IMOBA		√	√	√		√	√	√
IBEA			×	×			×	√
NSGA-II				×				√
SPEA2								
	UF7							
Method	IMOBA		IBEA		NSGA-II		SPEA2	
IMOBA			√		√		√	
IBEA					√		√	
NSGA-II							√	
SPEA2								

performance were found on these problems. Finally, Figure 6.6 represents the Pareto fronts obtained by IMOBA and NSGA-II in the median hypervolume executions for each test instance, along with the theoretical Pareto-optimal fronts.

The results obtained in these test problems confirm the relevance of IMOBA as a robust multiobjective method, obtaining significant results in both real-world problems like phylogenetic inference and complex multiobjective benchmarks. Attending to hypervolume performance, IMOBA is able to outperform IBEA, NSGA-II, and SPEA2 in all the considered scenarios. This statement is confirmed by Table 6.18, which points out the statistically significant improvement achieved by IMOBA over the reference methods. More specifically, IMOBA obtains average improvements of 5.39, 6.66, and 8.75 points in hypervolume with regard to IBEA, NSGA-II, and SPEA2, respectively, while also showing more stability in solution quality according to the reduced interquartile ranges.

By examining Figure 6.6, we can observe how IMOBA is able to generate satisfying approximations to the Pareto-optimal fronts in terms of convergence and diversity. Focusing on convergence,

the comparison with NSGA-II suggests that IMOBA obtains results of higher quality (according to their closeness to the Pareto-optimal solutions) in all the considered test instances. In fact, the representation in Figure 6.6 points out that a significant percentage of the solutions reported by the dominance-based reference algorithm are covered by IMOBA. Regarding the uniformity of the distributions observed in the attained outcomes, IMOBA also improves the diversity of the solutions contained in the median Pareto fronts in four of the problems under study. Therefore, this graphical comparison gives account of the high-quality approximations to the Pareto-optimal solutions achieved by IMOBA under different multiobjective contexts.

By extending the evaluation of multiobjective performance with these five CEC09 benchmark instances, the success of IMOBA in undertaking the solving of complex MOPs has been verified. In conclusion, our results reveal that an indicator-based swarm intelligence design provides suitable mechanisms to acquire advanced search capabilities in different multiobjective scenarios, improving the results reported by other well-established multiobjective methods.

6.3 Biological Performance Assessment

After conducting the multiobjective analysis of the applied metaheuristics, this section addresses now the assessment of biological quality of the inferred trees. With this purpose, we introduce firstly comparisons of parsimony and likelihood quality with regard to multi and single-criterion biological methods from the state-of-the-art, along with additional comparisons with other multiobjective solutions to phylogenetic inference proposed in the literature. Afterwards, the biological implications of multiobjective optimization are discussed in a real-world scenario by presenting and examining pure parsimony, pure likelihood, and multiobjective parsimony+likelihood phylogenies. Finally, consensus trees which summarize the phylogenies which compose the generated Pareto fronts are reported.

6.3.1 Biological Comparisons with Other Proposals

First of all, we proceed with the assessment of biological quality of the parsimony and likelihood trees reported by the applied multiobjective designs by making comparisons with other phylogenetic methods from the literature. As representative methods of each development line of multiobjective algorithms, our two best metaheuristics, IMOBA and MOABC, are considered for evaluation purposes. For each dataset, the phylogenies analyzed here correspond to the extreme points of the Pareto fronts inferred by each metaheuristic in the median hypervolume execution.

The study of the biological results retrieved by IMOBA and MOABC is conducted by making comparisons with a variety of biological methods. On the one hand, parsimony topologies are analyzed with regard to two pure parsimony methods: TNT [72] and DNAPARS [58]. On the other hand, likelihood results are assessed by taking as reference the solutions reported by five pure likelihood methods: RAxML [171], IQPNNI [120], MrBayes [148], Garli [200], and MetaPIGA [80]. In order to make fair comparisons, 31 runs per experiment with these tools were carried out under the

same experimental conditions as IMOBA and MOABC in the same hardware architecture. Source codes for DNAPARS, RAxML, IQPNNI, MrBayes, and Garli were compiled by using GCC with the -O3 optimization flag enabled, MetaPIGA was compiled by using JAVA 1.6.0_20, and the TNT executable file was downloaded from http://www.lillo.org.ar/phylogeny/tnt/. The configuration of input parameters was performed by considering the mean execution times shown by our best overall design (IMOBA) on each dataset (using 12 cores, bearing in mind that these biological methods have multicore capabilities). As parsimony computations require less execution time than the likelihood ones, the parsimony-based methods were configured attending to the times spent by IMOBA on parsimony operations. Table 6.19 reports the input parameters for each method. To carry out this comparison, we have used the median parsimony/likelihood trees for each method.

The upper side of Table 6.20 compares the parsimony scores reported by IMOBA, MOABC, TNT, and DNAPARS. This table also shows the results of applying a statistical testing of the examined trees under the Shimodaira-Hasegawa SH test [105], which searches for statistically significant differences in tree topologies according to the parsimony criterion. This table points out the meaningful parsimony results achieved by our multiobjective metaheuristics, matching the parsimony values provided by the well-known tool TNT in all the data sets and also improving the scores from DNAPARS in four data sets (*HIV1_192*, *RDPII_218*, *S1482_346*, and *ZILLA_500*). Furthermore, the results of applying the SH test for parsimony show that IMOBA and MOABC are able to obtain parsimony topologies which are statistically comparable to the solutions reported by pure parsimony methods. Therefore, the parsimony comparison suggests the success of these multiobjective designs in finding high-quality phylogenies from a parsimony perspective.

With regard to the likelihood criterion, the bottom side of Table 6.20 introduces the comparison of likelihood results obtained by IMOBA, MOABC, RAxML, IQPNNI, MrBayes, Garli, and MetaPIGA. Due to the fact that comparisons based on the likelihood scores reported by each tool separately are not reliable due to different implementations of the likelihood function, we aim to conduct a standardized comparison by reporting the likelihood scores obtained when using BIO++ to evaluate each tree. Moreover, in order to assess likelihood performance under a statistically reliable methodology, we have carried out the statistical testing of likelihood trees by using CONSEL approximately unbiased AU tests [159]. As explained in Chapter 4, the AU values reported by CONSEL classify the examined topologies in accordance with their quality in such a way that higher scores denote higher statistical chances of achieving the best likelihood values. From a likelihood point of view, it can be observed the significant quality of the likelihood trees obtained by IMOBA, which attains the best likelihood scores on *HIV1_192*, *RDPII_218*, and *S1482_346*. This statement is supported by CONSEL, which assigns the highest AU values to the solutions reported by this metaheuristic for these three data sets. The bat-inspired design is also able to improve the results obtained by the biological methods for the dataset with the highest number of species *ZILLA_500*, generating the second best likelihood solution. MOABC also shows significant performance from a likelihood perspective, achieving the best likelihood tree attending to its score and CONSEL value on *ZILLA_500*, while reporting the second best values on *rbcL_55*, *HIV1_192*, and *S1482_346*.

Table 6.19 Biological methods - input settings

	rbcL_55
TNT	Sectorial search, tree fusing, stop time = 129.16s
DNAPARS	More thorough search, 100 trees saved, randomize input 300 times
RAxML	Default settings, number of trees to be generated = 75
IQPNNI	Default settings, number of generations = 1500
MrBayes	Number of generations = 32000, nruns = 1, nchains = 48
Garli	Population size = 100, stop time = 543.70s
MetaPIGA	Num. populations = 12, pop. size = 8, stop time = 543.70s
	mtDNA_186
TNT	Sectorial search, tree fusing, stop time = 1763.73s
DNAPARS	More thorough search, 100 trees saved, randomize input 100 times
RAxML	Default settings, number of trees to be generated = 90
IQPNNI	Default settings, number of generations = 3000
MrBayes	Number of generations = 25000, nruns = 1, nchains = 48
Garli	Population size = 100, stop time = 4255.21s
MetaPIGA	Num. populations = 12, pop. size = 8, stop time = 4255.21s
	HIV1_192
TNT	Sectorial search, tree fusing, stop time = 490.10s
DNAPARS	More thorough search, 100 trees saved, randomize input 50 times
RAxML	Default settings, number of trees to be generated = 20
IQPNNI	Default settings, number of generations = 750
MrBayes	Number of generations = 42000, nruns = 1, nchains = 48
Garli	Population size = 100, stop time = 1529.41s
MetaPIGA	Num. populations = 12, pop. size = 8, stop time = 1529.41s
	RDPII_218
TNT	Sectorial search, tree fusing, stop time = 1551.23s
DNAPARS	More thorough search, 100 trees saved, randomize input 40 times
RAxML	Default settings, number of trees to be generated = 20
IQPNNI	Default settings, number of generations = 825
MrBayes	Number of generations = 42500, nruns = 1, nchains = 48
Garli	Population size = 100, stop time = 4940.22s
MetaPIGA	Num. populations = 12, pop. size = 8, stop time = 4940.22s
	S1482_346
TNT	Sectorial search, tree fusing, stop time = 1264.93s
DNAPARS	More thorough search, 100 trees saved, randomize input 60 times
RAxML	Default settings, number of trees to be generated = 45
IQPNNI	Default settings, number of generations = 550
MrBayes	Number of generations = 55000, nruns = 1, nchains = 48
Garli	Population size = 100, stop time = 3426.81s
MetaPIGA	Num. populations = 12, pop. size = 8, stop time = 3426.81s
	ZILLA_500
TNT	Sectorial search, tree fusing, stop time = 2724.54s
DNAPARS	More thorough search, 100 trees saved, randomize input 10 times
RAxML	Default settings, number of trees to be generated = 30
IQPNNI	Default settings, number of generations = 1500
MrBayes	Number of generations = 73000, nruns = 1, nchains = 48
Garli	Population size = 100, stop time = 6540.22s
MetaPIGA	Num. populations = 12, pop. size = 8, stop time = 6540.22s

Table 6.20 Parsimony and likelihood comparisons

Parsimony assessment						
	rbcL_55		*mtDNA_186*		*HIV1_192*	
Method	Parsimony score	SH P-value	Parsimony score	SH P-value	Parsimony score	SH P-value
IMOBA	**4874**	-best-	**2431**	-best-	**6091**	-best-
MOABC	**4874**	0.703	**2431**	0.666	**6091**	0.701
TNT	**4874**	0.578	**2431**	0.713	**6091**	0.733
DNAPARS	**4874**	0.519	**2431**	0.621	6104	0.330
	RDPII_218		*S1482_346*		*ZILLA_500*	
IMOBA	**41488**	-best-	**2058**	-best-	**16218**	-best-
MOABC	**41488**	0.688	**2058**	0.641	**16218**	0.677
TNT	**41488**	0.689	**2058**	0.688	**16218**	0.688
DNAPARS	41587	0.052	2060	0.507	16224	0.458
Likelihood assessment						
	rbcL_55		*mtDNA_186*		*HIV1_192*	
Method	Likelihood score	AU value	Likelihood score	AU value	Likelihood score	AU value
IMOBA	-21781.736	0.399	-39868.609	0.382	**-21656.682**	**0.748**
MOABC	-21778.547	0.562	-39868.666	0.381	-21662.503	0.510
RAxML	-21788.123	0.191	-39868.910	0.375	-21672.805	0.473
IQPNNI	-21787.127	0.146	**-39865.061**	**0.691**	-21688.703	0.332
MrBayes	-21824.425	0.001	-48693.892	0.000	-22361.030	0.000
Garli	**-21770.026**	**0.664**	-39868.025	0.386	-21696.444	0.279
MetaPIGA	-21827.173	0.000	-41957.399	0.000	-23388.549	0.000
	RDPII_218		*S1482_346*		*ZILLA_500*	
IMOBA	**-134070.980**	**0.690**	**-9608.028**	**0.665**	-80566.194	0.653
MOABC	-134075.884	0.463	-9612.010	0.633	**-80565.429**	**0.676**
RAxML	-134096.964	0.348	-9615.009	0.399	-80598.900	0.213
IQPNNI	-134128.005	0.180	-9637.472	0.194	-80610.941	0.146
MrBayes	-137422.355	0.000	-15377.699	0.000	-96928.394	0.001
Garli	-134074.663	0.555	-9630.034	0.231	-80578.855	0.332
MetaPIGA	-137689.363	0.000	-23236.960	0.000	-112710.786	0.000

Table 6.21 Biological comparisons with PhyloMOEA (HKY85+Γ model)

	Parsimony score			Likelihood score		
Dataset	IMOBA	MOABC	PhyloMOEA	IMOBA	MOABC	PhyloMOEA
rbcL_55	**4874**	**4874**	**4874**	-21812.42	**-21811.53**	-21889.84
mtDNA_186	**2431**	**2431**	2437	**-39888.43**	-39889.28	-39896.44
RDPII_218	**41488**	**41488**	41534	**-134151.08**	-134177.87	-134696.53
ZILLA_500	**16218**	**16218**	16219	**-80965.73**	-80968.76	-81018.06

Next, we examine in Table 6.21 the quality of the trees reported by IMOBA and MOABC with regard to PhyloMOEA, the multiobjective proposal published by Cancino and Delbem to conduct multiobjective phylogenetic analyses attending to parsimony and likelihood [24]. The algorithmic

design of PhyloMOEA is based on NSGA-II, considering a tree-shaped individual representation and evolutionary operators which involve the application of PDG and NNI movements for crossover and mutation purposes. The results reported by this tool in [24] were obtained by performing phylogenetic searches under the HKY85 evolutionary model [105] over the data sets *rbcL_55*, *mtDNA_186*, *RDPII_218*, and *ZILLA_500*. With the aim of making fair comparisons, Table 6.21 reports the results obtained by using IMOBA and MOABC to carry out 31 runs per experiment over these instances under this model of nucleotide evolution. According to our experimental results, our proposals obtain significant performance attending to both parsimony and likelihood perspectives, outperforming PhyloMOEA in all the analyzed data sets. From a parsimony perspective, both IMOBA and MOABC succeeds in reporting improved parsimony trees on *mtDNA_186*, *RDPII_218*, and *ZILLA_500*. Attending to likelihood results, IMOBA gives rise to the best solutions in three of the considered data sets, while the remaining one is successfully addressed by MOABC.

Therefore, these comparisons have given account of the significant biological quality shown by the solutions generated by our two best multiobjective metaheuristics, IMOBA and MOABC. As in the multiobjective performance comparison, IMOBA goes a step further over MOABC, although both algorithms are able to achieve high-quality phylogenies from a parsimony and likelihood perspective with regard to the state-of-the-art methods under evaluation. These results support the statements discussed in the multiobjective analysis, showing the relevance of applying advanced swarm intelligence search engines to address this kind of challenging biological problems.

6.3.2 Biological Implications

In Chapter 2, the main goals behind the formulation of phylogenetic inference as a MOP were defined. The first one involved the requirement of generating in a single run high-quality phylogenies attending to each considered objective separately. The results previously analyzed suggest the success of multiobjective algorithms in providing satisfying maximum parsimony and maximum likelihood phylogenetic hypotheses with regard to the state-of-the-art. Now we undertake the discussion of the second main goal: the usefulness of the non-extreme solutions contained in the Pareto set. According to the formulation, multiobjective phylogenetic topologies can provide relevant information from two points of view. Firstly, such multiobjective hypotheses represent a compromise answer to solve incongruence issues due to optimality criteria, considering evolutionary relationships with biological sense attending to parsimony and likelihood which are not successfully recovered by single-criterion trees separately. Secondly, consensus methods [19] can be applied to generate a representative phylogenetic tree which combines the information contained in all the Pareto front, introducing more robustness from a statistical and biological point of view. In the following subsections, multiobjective optimization techniques are applied to solve a well-known real scenario where the literature has reported conflicting hypotheses according to the used optimality criterion (parsimony or likelihood). In addition, we present the consensus trees generated in our experimentation for the different data sets used in the study of multiobjective performance.

Multiobjective Phylogeny of Plethodontid Salamanders

In order to explore the relevance of multiobjective optimization to address incongruence issues in phylogenetics, we present a case study involving the analysis of the *M2771_27* dataset. This instance contains 27 sequences of 15778 nucleotides corresponding to Plethodontid salamander mitochondrial DNA data [124]. Multiple researches have studied the phylogeny of these organisms under parsimony and likelihood, showing significant conflicts in the evolutionary relationships inferred by each method (as published in [111], [185], and [32]). Multiobjective optimization techniques have been applied to verify if the multiobjective formulation of the problem is able to generate an accurate compromise answer to the incongruence observed in the single-criterion hypotheses. For this purpose, NSGA-II was applied to analyze this dataset. In spite of not being the leading algorithm in our comparisons, the choice of NSGA-II was done due to the fact that this metaheuristic represents a well-established reference algorithm in evolutionary multiobjective optimization.

Experiments with NSGA-II over this dataset were performed under the evolutionary model GTR+Γ, according to the model selection conducted by jModelTest. Figure 6.7 shows the multiobjective topology generated by NSGA-II which contributed most to the overall hypervolume obtained in the execution which scored the median value of this indicator. On the other hand, Figures 6.8 and 6.9 show the consensus parsimony and likelihood hypotheses for Plethodontid salamander evolution, generated from 100 bootstrap samples. By examining Figure 6.7, it can be observed that the multiobjective tree is able to identify the major subfamilies in the Plethodontid salamander taxonomy [111]. The species *Plethodon elongatus*, *Plethodon cinereus*, *Plethodon petraeus*, *Phaeognathus hubrichti*, *Desmognathus wrighti*, *Desmognathus fuscus*, *Aneides flavipunctatus*, *Aneides hardii*, *Ensantina eschscholtzii*, *Hydromantes italicus*, and *Hydromantes brunus* are included in the family Plethodontinae. The families Hemidactylinae, Bolitoglossinae, and Spelerpinae are also identified. In Hemidactylinae/Bolitoglossinae, we can find the species *Hemidactylium scutatum*, *Batrachoseps wrighti*, *Batrachoseps attenuatus*, *Oedipina poelzi*, *Nototriton abscondens*, *Thorius suchi*, and *Bolitoglossa sombra*. On the other hand, *Eurycea bislineata*, *Pseudotriton ruber*, *Gyrinophilus porphyriticus*, and *Stereochilus marginatus* are represented in Spelerpinae. Finally, those species which belong to other major genera *Andrias davidianus* (Cryptobranchidae), *Ranodon sibiricus* (Hynobiidae), *Ambystoma laterale* (Ambystomatidae), *Mertensiella luschani* (Salamandrinae), and *Rhyacotriton variegatus* (Rhyacotritonidae) are included in a separate clade.

By comparing the multiobjective tree with the ones reported by the maximum parsimony and maximum likelihood approaches, some relationships with a significant biological relevance can be identified. On the one hand, the position of *Hemidactylium scutatum* represents a noticeable source of conflict between parsimony and likelihood. The research carried out in [185] suggests the inclusion of this organism into the Hemidactylinae family and its relationship with the Batrachoseps subclade. This statement is successfully supported by our multiobjective tree and the maximum likelihood topology. On the other hand, parsimonious relationships are also found in the multiobjective hypothesis. The parsimony and multiobjective phylogenies successfully describe the monophyly be-

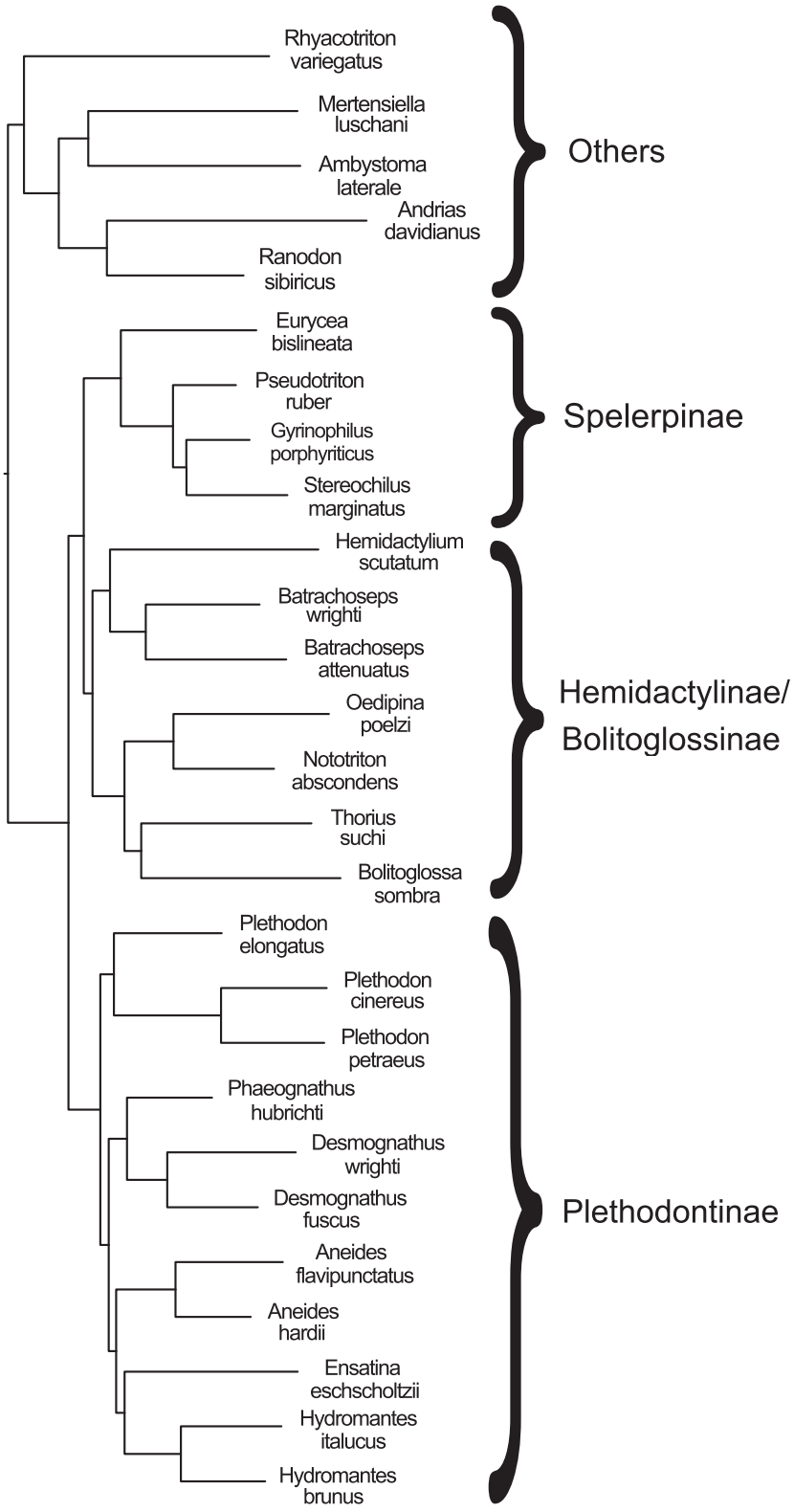


Fig. 6.7 Multiobjective phylogeny for Plethodontid salamander mitochondrial DNA

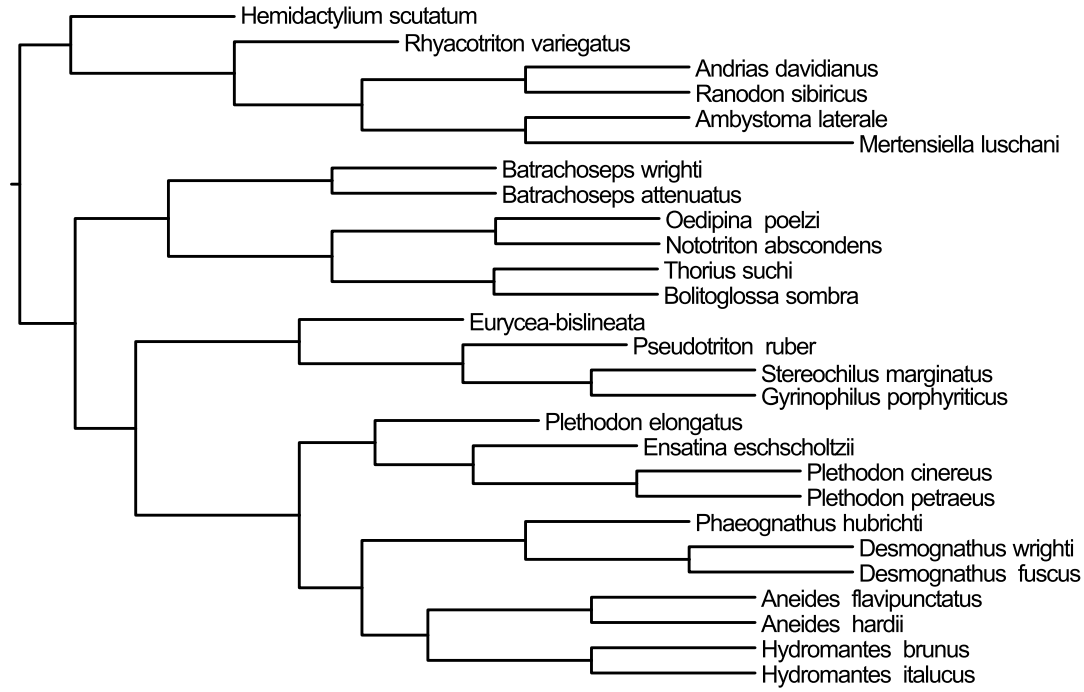


Fig. 6.8 Maximum parsimony phylogeny for Plethodontid salamander mitochondrial DNA

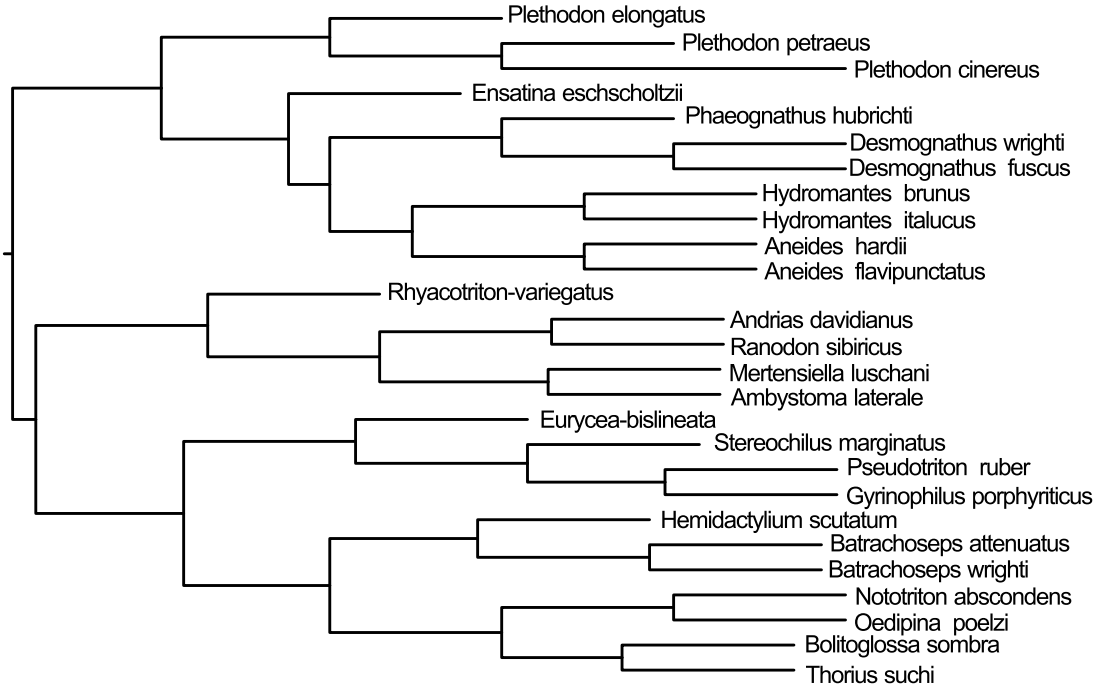


Fig. 6.9 Maximum likelihood phylogeny for Plethodontid salamander mitochondrial DNA

tween *Gyrinophilus porphyriticus* and *Stereochilus marginatus* as described in [32] and [111], while the likelihood tree introduces *Pseudotriton ruber* as the sister taxon of *Gyrinophilus*. These two examples show that the examined multiobjective topology represents a satisfying compromise hypothesis in accordance with the biological taxonomy of these organisms, dealing with the conflicts reported by single-criterion parsimony and likelihood trees.

In conclusion, this case study suggests the biological relevance of providing phylogenetic explanations under a multiobjective formulation of the problem. The analyzed multiobjective topology was able to solve the conflicting relationships inferred by single-criterion analyses, while satisfactorily identifying major salamander families in accordance with the taxonomy of these organisms.

Consensus Trees

Next, consensus trees for the different alignments examined in this research are reported. The idea is to combine the knowledge provided by all the inferred topologies by applying a consensus method which takes as input the Pareto set to generate a representative phylogenetic hypothesis for a given dataset. In this way, consensus phylogenies are able to shed light on the most supported evolutionary relationships observed in the attained solutions. The consensus method considered in this research is the extended majority rule approach implemented in the CONSENSE application from the PHYLIP tools package [58]. Given a set of Pareto solutions, this method generates a consensus tree by including those sets of species which appear in more than 50% of the input trees, considering afterwards the remaining relationships in order of appearance frequency until the topology is fully resolved. The phylogenies which have been used to carry out this analysis are the ones inferred by our best multiobjective bioinspired metaheuristic, IMOBA.

The consensus phylogenetic trees generated under this methodology are represented in Figures 6.10 (for *rbcL_55*), 6.11 - 6.12 (*mtDNA_186*), 6.13 - 6.14 (*HIV1_192*), and 6.15 - 6.16 (*RDPII_218*). Due to their large size, the consensus phylogenies obtained for *S1482_346* and *ZILLA_500* are available for visualization purposes at http://arco.unex.es/sesaji/maintrees.htm. This website also contains a graphical representation of descriptive phylogenetic trees inferred by IMOBA for each dataset. These representative phylogenies include the maximum parsimony and likelihood topologies, the consensus tree, and an example of multiobjective phylogenetic hypothesis selected by using the L2-metric, a multiobjective decision making method which gives preference to the solution in the Pareto front that minimizes the Euclidean distance to the ideal reference point.

Concluding Remarks

The study of biological implications gives account of the new opportunities that imply this novel multiobjective perspective of phylogenetic inference. By designing multiobjective approaches, biologists will count with accurate tools to generate in a single run high-quality phylogenies from a biological point of view, as reported in the comparison with other single-criterion and multiobjective methods from the state-of-the-art. Along with this, a multiobjective approach will lead to the

reconstruction of a set of Pareto solutions which suggest alternative hypotheses attending to multiple criteria simultaneously. The successful solving of incongruence issues in the phylogeny of Plethodontid salamanders shows that these multiobjective hypotheses are useful to address the conflicts which arise when parsimony and likelihood disagree about the phylogenetic relationships which best represent the evolution of the input data. In addition, by providing a wide variety of alternative hypotheses in the Pareto set, the expert receives additional information on the conflicts observed between objectives and the characteristics of the input alignment, along with having the opportunity to apply decision making techniques to choose the most satisfying phylogeny. Finally, the application of multiobjective optimization techniques to phylogenetics represents a promising approach to introduce more robustness to the inference process, allowing biologists to obtain more reliable conclusions from a biostatistical point of view by using consensus methods which summarize the most supported evolutionary relationships observed in the Pareto set.

6.4 Summary

This chapter has undertaken the multiobjective and biological evaluation of the bioinspired metaheuristics applied in this research to conduct phylogenetic analyses. In a first step, we have discussed the evolutionary framework in which this analysis was conducted, identifying the best model of nucleotide evolution for the six real data sets considered in our experimentation. In addition, details on the computation of the multiobjective metrics used for performance evaluation in this comparative study (hypervolume, coverage relation, and spacing) have been described. After that, some design choices included in the implementations of the metaheuristics have been discussed. Firstly, the consideration of a distance-based methodology to carry out phylogenetic searches has been justified by observing the relative multiobjective performance attained with regard to traditional tree-based engines, observing statistically significant improvements in the quality of the inferred Pareto approximation sets. Secondly, the choice of BIONJ as tree-building method has been examined by making comparisons with other distance-based methods. Our results revealed the accuracy of BIONJ when making accurate mappings from the distance matrix space to the phylogenetic tree space, outperforming clustering methods in all the considered scenarios.

After discussing these initial considerations, the multiobjective performance analysis of the applied metaheuristics was carried out. The main goal of this study laid on the comparison of different algorithmic designs, defined in accordance with two of the most popular development lines in multiobjective optimization: dominance-based and indicator-based approaches. By considering each line separately, we have given account of the relevance of applying swarm intelligence to address this complex biological problem. On the dominance-based side, the most satisfying metaheuristic was MOABC, which achieved accurate performance attending to hypervolume, spacing, and set coverage, outperforming MO-FA and the reference MOEAs, NSGA-II and SPEA2. On the indicator-based side, IMOBA reported improved results over the well-known IBEA, confirming the implications of swarm intelligence as a useful tool to deal with the phylogenetic reconstruction problem. Com-

parisons between the two lines of algorithmic development have pointed out the relevance of the indicator-based designs. Focusing on swarm intelligence, IMOBA was able to obtain a statistically significant improvement over MOABC and MO-FA in almost all the considered scenarios. In addition, IBEA reported significant results in comparison to NSGA-II and SPEA2. In overall, the metaheuristic which showed the best performance from a multiobjective perspective was IMOBA, suggesting that the combination of quality indicators-based searches and swarm intelligence techniques represents a promising approach to tackle this kind of challenging real-world MOPs.

Finally, the biological results reported by our two best multiobjective designs, IMOBA (indicator side) and MOABC (dominance side) have been examined. For this purpose, comparisons with up to eight biological methods from the literature were conducted. The evaluation of parsimony scores under the SH tests has shown how both IMOBA and MOABC were able to match the parsimony quality of the trees obtained by the well-known tool TNT, generating statistically comparable results with regard to this state-of-the-art method while improving the scores reported by the classic method DNAPARS. In addition, both metaheuristics were also able to obtain significant likelihood results in comparison to widely-used methods, such as RAxML, Garli, IQPNNI, MrBayes, and MetaPIGA. In summary, the likelihood trees reported by IMOBA and MOABC attained the best likelihood scores in four of the considered data sets, results which were supported by the biostatistical comparison of likelihood solutions under the CONSEL AU tests. The quality of the inferred trees was also confirmed by the comparison with the multiobjective proposal PhyloMOEA. Afterwards, the study of phylogenetic results proceeded with the study of the biological implications associated to the application of multiobjective optimization techniques to this problem. By applying a multiobjective approach, we have shown how multiobjective topologies are able to provide compromise answers to the conflicts observed between pure parsimony and likelihood hypotheses on a widely-studied alignment of Plethodontid salamander DNA data. Finally, consensus trees generated by combining the information provided by the entire Pareto front have been reported for the different data sets considered in this study. In conclusion, multiobjective optimization represents a promising tool to address the phylogeny inference problem, generating high-quality phylogenetic trees attending to parsimony and likelihood in a single run while providing a set of Pareto solutions to address incongruence issues by optimizing both criteria simultaneously.

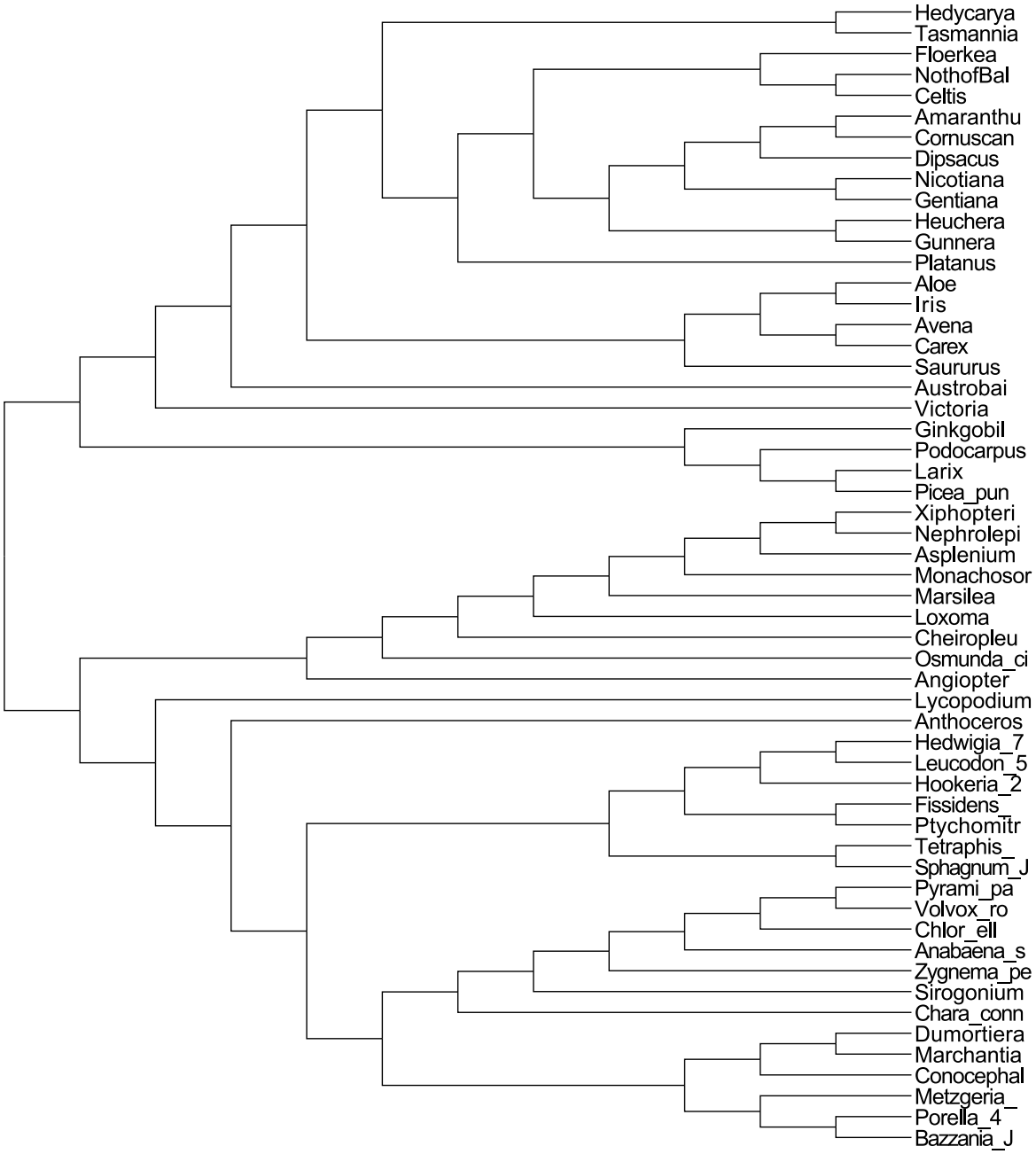


Fig. 6.10 Consensus topology for *rbcL_55*

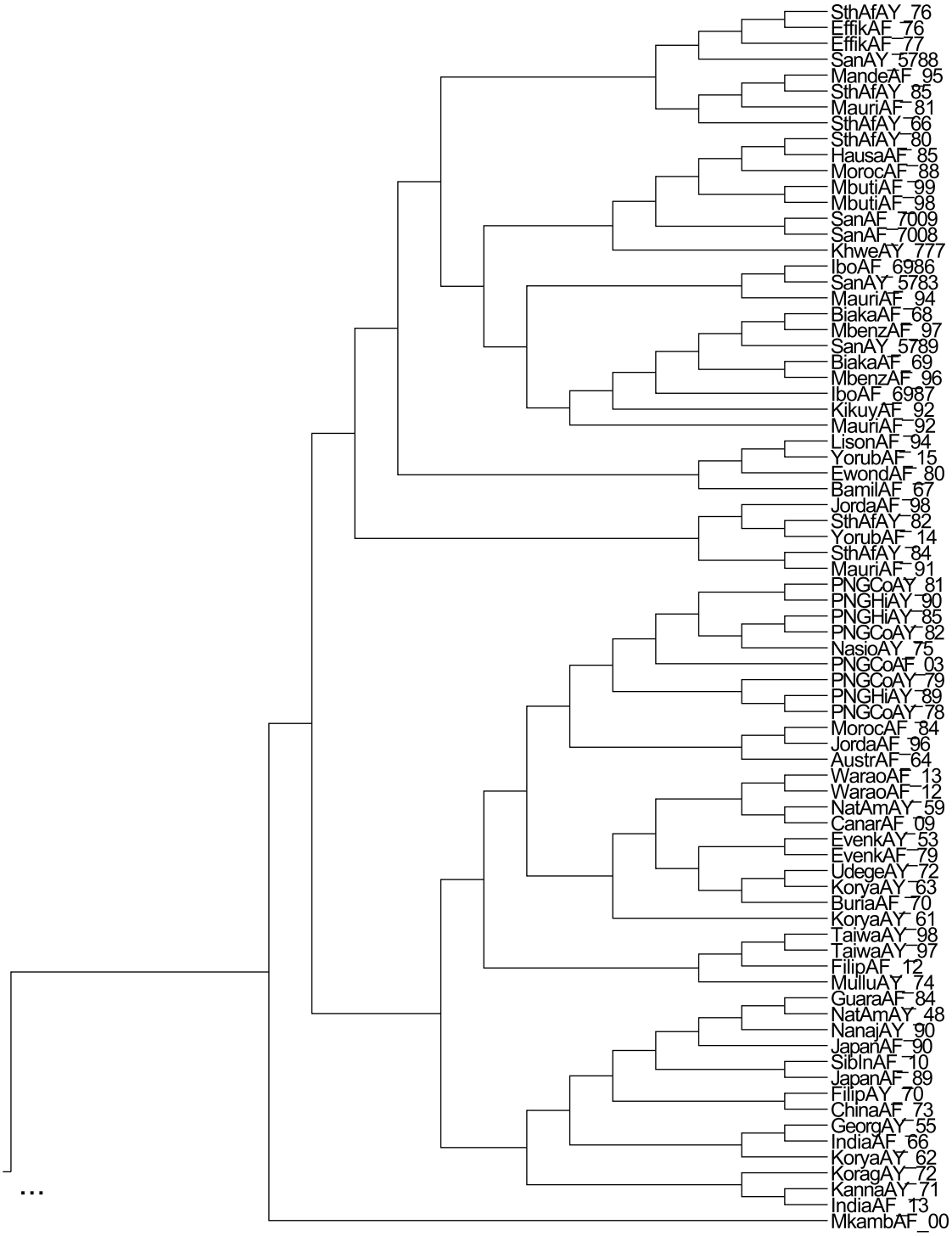


Fig. 6.11 Consensus topology for *mtDNA_186* (1/2)

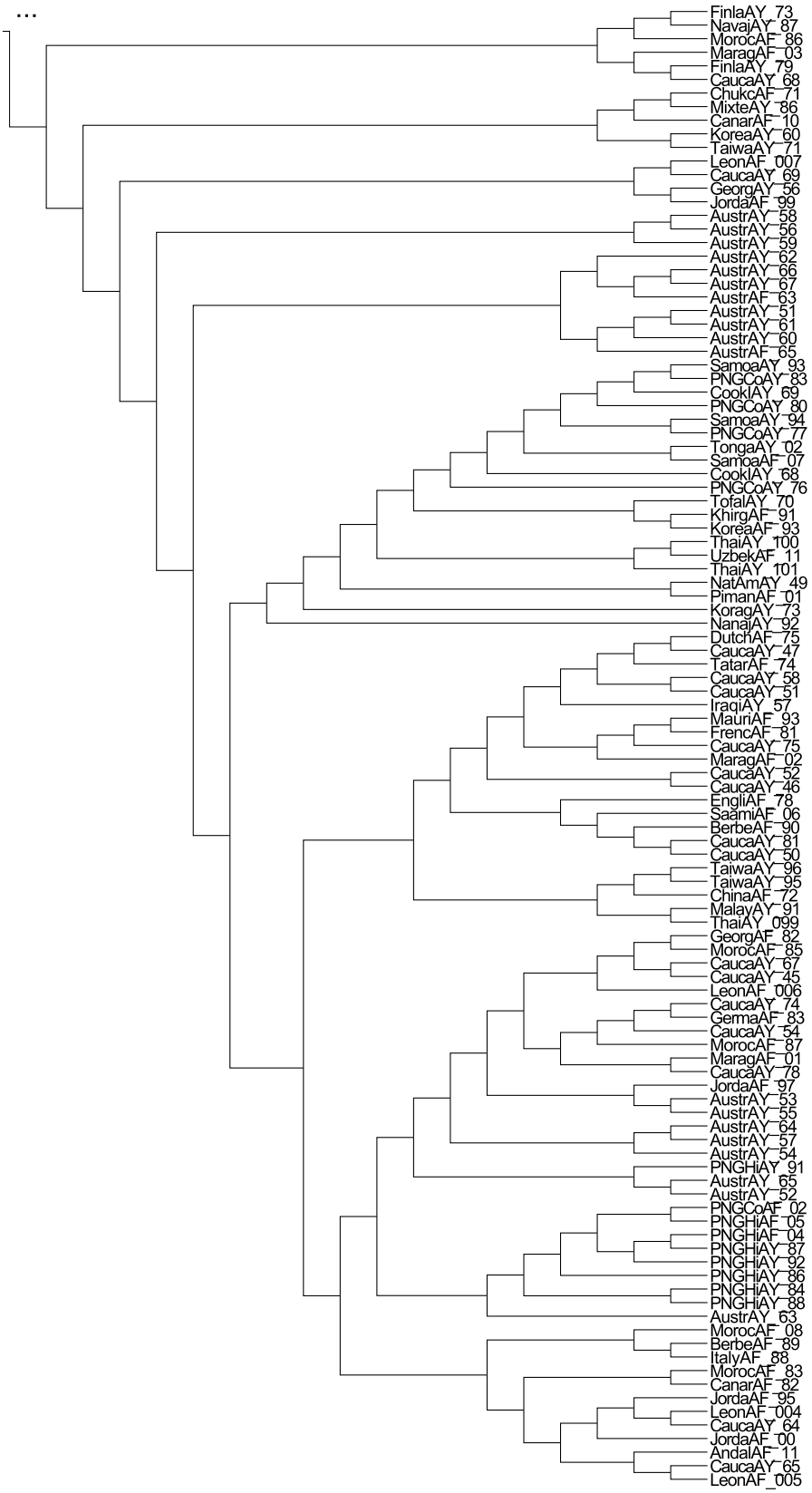


Fig. 6.12 Consensus topology for *mtDNA_186* (2/2)

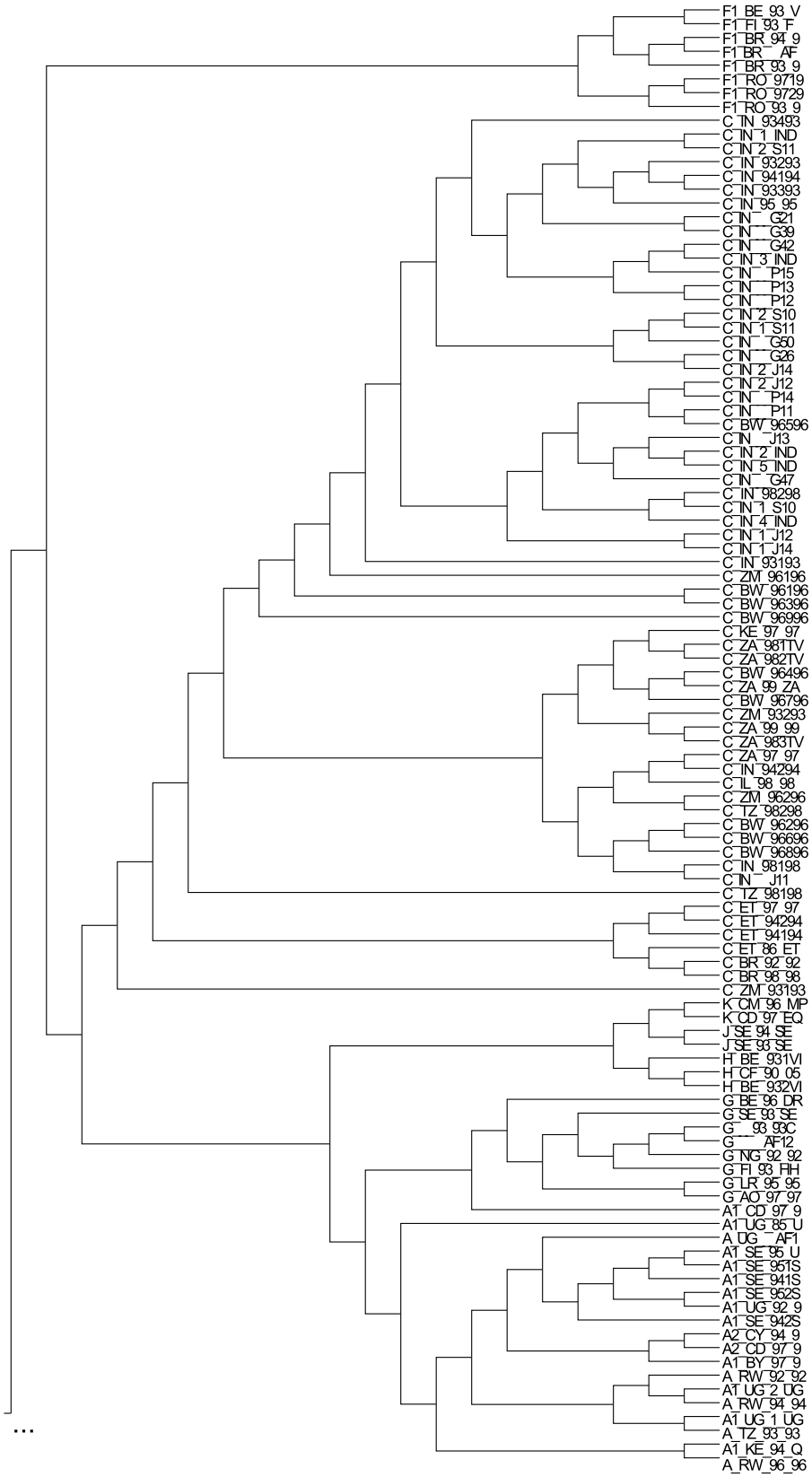


Fig. 6.13 Consensus topology for *HIV1_192* (1/2)

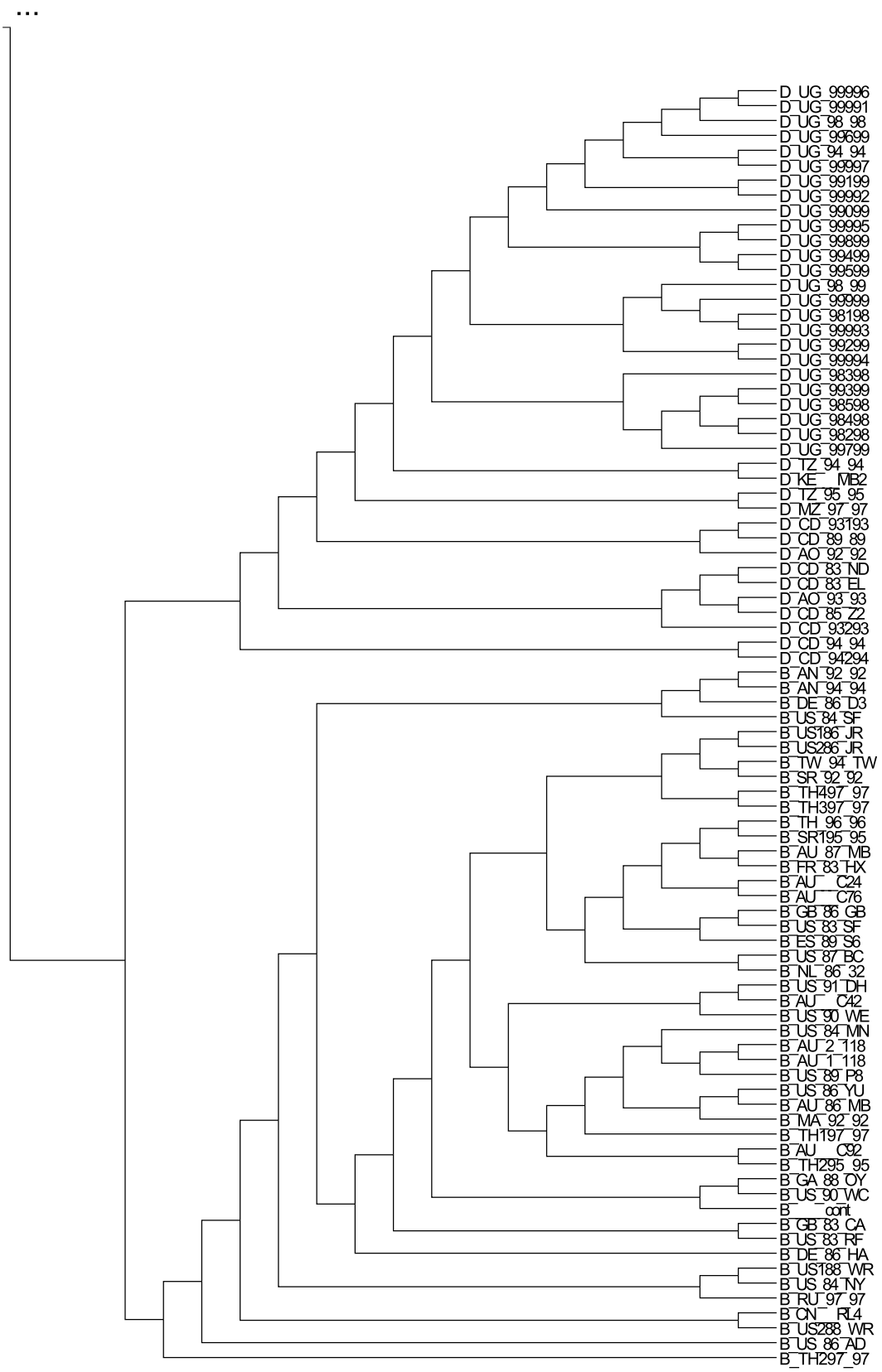


Fig. 6.14 Consensus topology for *HIV1_192* (2/2)

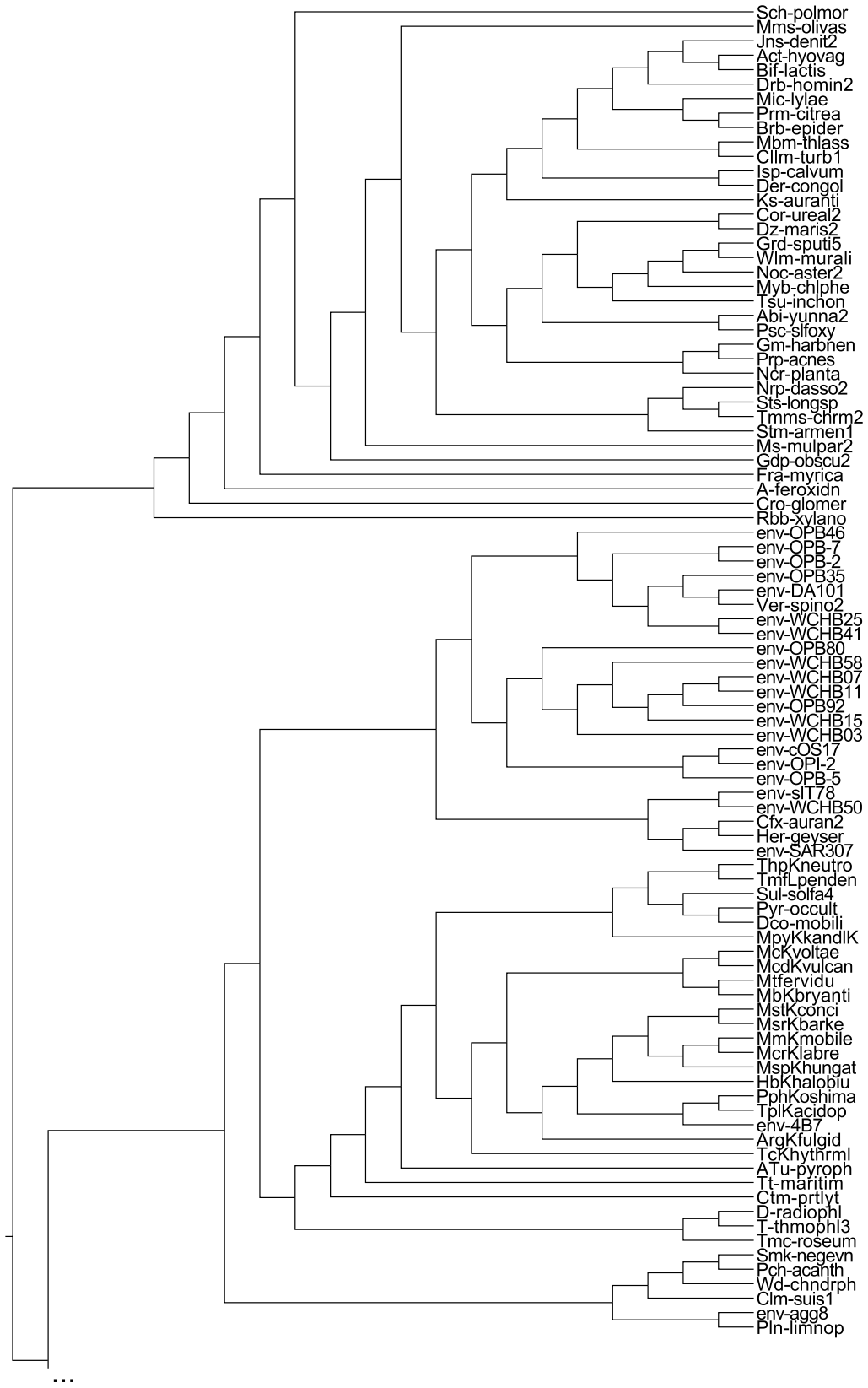


Fig. 6.15 Consensus topology for *RDPII_218* (1/2)

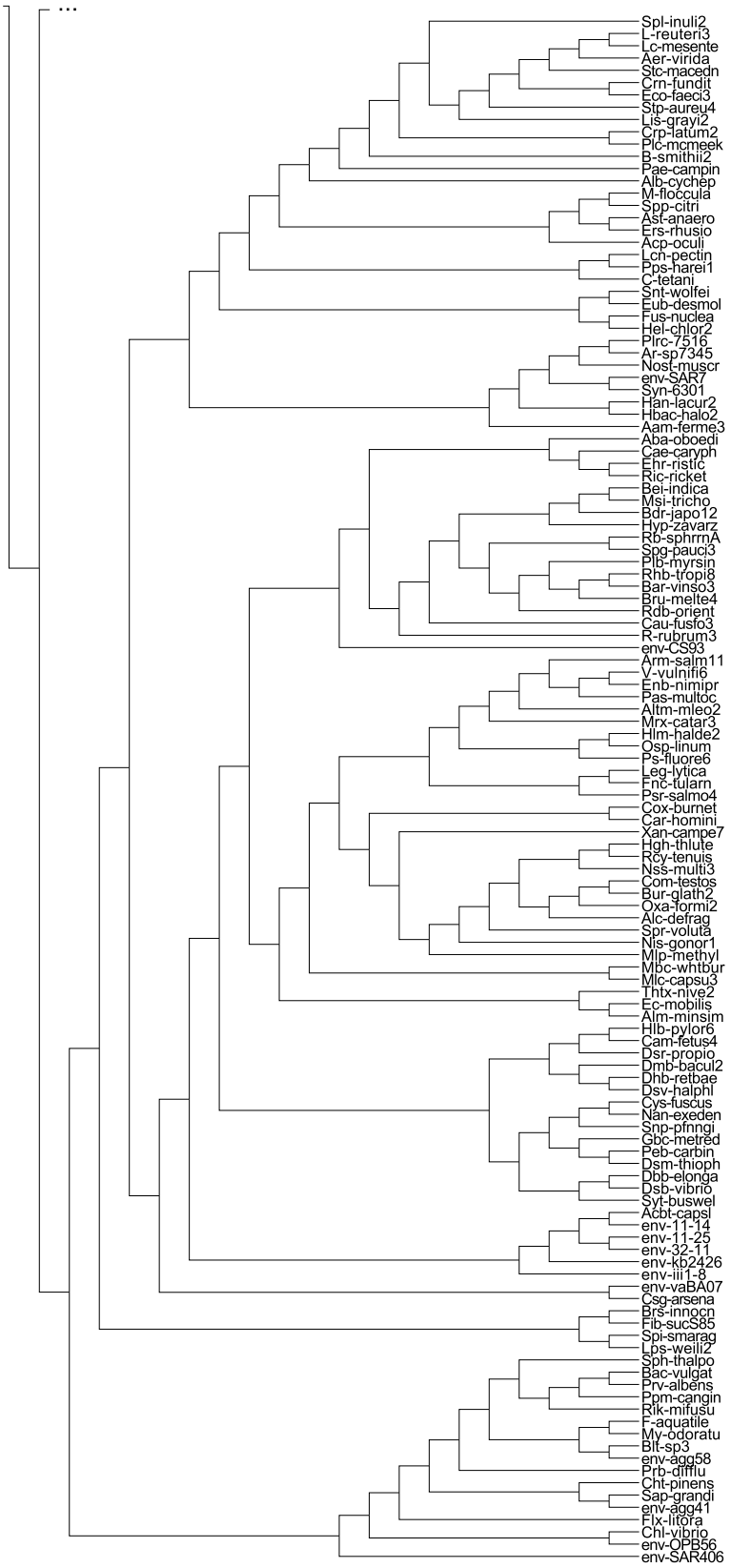


Fig. 6.16 Consensus topology for *RDPII_218* (2/2)

Chapter 7

Metaheuristic Parallelization Approaches

This chapter deals with the parallelization of multiobjective metaheuristics for phylogenetics. The main goal is to take advantage of the opportunities provided by the combination of bioinspired computing and parallelism in order to address time-consuming problems. For this purpose, parallel designs for two main hardware configurations are proposed. Firstly, parallel metaheuristic designs based on OpenMP are described to exploit the capabilities of shared-memory multicore multiprocessor systems. In this sense, two main parallelization strategies are examined: synchronous generational designs and asynchronous non-generational approaches. Secondly, mixed mode programming techniques based on MPI+OpenMP are studied to parallelize bioinspired metaheuristics on hybrid shared-distributed memory clusters.

7.1 On the Parallelization of Multiobjective Metaheuristics

This section provides hints on the parallelization of bioinspired metaheuristics, analyzing the time profile of the algorithms studied in this Thesis to accurately integrate parallelism into their designs. Multiobjective metaheuristics aim to find a Pareto set containing solutions which optimize two or more objective functions simultaneously. In real-world scenarios, the solving of such problems represents a significant challenge, as the process of finding a good approximation to the Pareto-optimal set becomes very computationally demanding. When dealing with MOPs with large search spaces and time-consuming objective functions, the efforts of the developer must be focused on proposing efficient and effective designs to address the computational complexity of the problem.

One of the most important features of bioinspired algorithms is their suitability to be improved by means of parallel computing. These methods often involve a population of solutions whose evolution can be carried out by mapping the computation of evolutionary operators and fitness evaluations to different parallel processes. Throughout the years, major research works have discussed the inherent

machine-independent parallelism of representative bioinspired designs, such as genetic algorithms [73], pointing out the implicit parallelism shown by these algorithmic approaches [21].

This parallel nature has motivated the development of parallel metaheuristics to solve NP-hard problems. As described in Chapter 4 - Section 4.2.8, parallel computing techniques can be integrated into multiobjective optimizers to decompose the problem and decrease execution time by distributing independent tasks among multiple processors. Parallelism may also be applied to improve the quality of the results, by allowing the metaheuristic to explore more regions of the solution space in the same amount of time as the serial counterpart. In this Thesis, we are interested in exploring the first line of parallel developments, often classified as intra-algorithm parallelization schemes, due to the fact that phylogenetic inference represents a computationally demanding optimization problem. That is, the computing capabilities of current hardware architectures will be exploited with the aim of reducing the execution time required to complete multiobjective phylogenetic analyses on real data sets. For this purpose, different parallel programming paradigms can be applied. In this chapter, we explore problem independent parallel designs for our metaheuristics, proposing OpenMP-based designs for shared-memory environments and cluster computing techniques based on hybrid MPI+OpenMP schemes for shared-distributed memory platforms. On the other hand, Chapters 9 and 10 will deal with the study of problem dependent parallel designs, reporting as case study the fine-grained parallelization of objective functions by means of heterogeneous computing.

7.1.1 Execution Profile Analysis

In order to develop efficient parallel approaches and understand how the studied methods can benefit from the exploitation of parallel architectures, the first step to be considered is the analysis of the application time profile. Such study aims to check the way the operations included in our metaheuristics contribute to the overall execution time, identifying the most time-consuming sets of instructions which should be addressed by our parallelization efforts. To this end, we have examined the profiles of the serial versions of NSGA-II (as representative example of evolutionary algorithms) and MO-FA (representing swarm intelligence). Figure 7.1 shows the average percentage of execution time spent by NSGA-II and MO-FA on computing evolutionary operations (which include selection, crossover, and mutation operators in NSGA-II and firefly movements in MO-FA), tree-building operations (via BIONJ), topology optimizations, the evaluation of solutions attending to parsimony and likelihood, and other operations (related to the population and Pareto set data structures management).

The profile of NSGA-II and MO-FA reported that the evaluation of candidate solutions under parsimony and likelihood represented the most computationally demanding operation in both metaheuristics, contributing percentages over 80.3% and 78.5%, respectively, of their overall execution times. The BIONJ-based reconstruction of trees from distance matrices and the topological optimization procedure also have a noticeable impact in processing times. More specifically, we observe time percentages of 4.9% (NSGA-II) / 5.1% (MO-FA) which are spent on the BIONJ tree-building method and percentages of 13.5% (NSGA-II) / 14.4% (MO-FA) spent on topological searches and

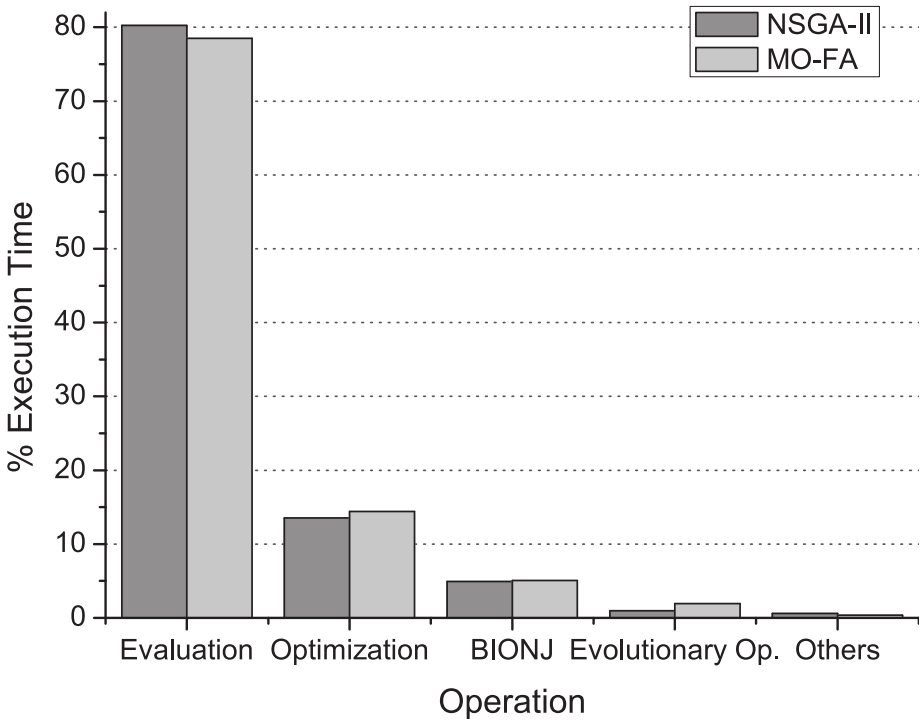


Fig. 7.1 Serial time profile analysis of NSGA-II and MO-FA, showing the contribution of each operation to the overall execution time

optimizations. Finally, evolutionary and data structure management operations contribute about a 2% of the overall execution time in both algorithms, being more significant the times required by the firefly movements in MO-FA than the ones spent on evolutionary operators in NSGA-II.

Once the most time-demanding operations have been identified, their suitability to be parallelized must be verified. As initially remarked, the main parallel opportunities are given by the fact that the algorithms studied in this Thesis consider a population-based design, operating their loops over a data structure composed of multiple candidate solutions to the problem. The most consuming operations according to the profiles are the ones related to the inference and evaluation of phylogenetic trees. These instructions are generally enclosed in loops which operate over each individual in the population/swarm, in such a way that the i-th iteration processes the distance matrix generation, tree inference, and evaluation for an individual i, showing no dependencies with regard to the remaining iterations (individuals). This consideration leads to the definition of independent tasks that can be processed in parallel with the aim of minimizing computing time. It should be also noted that these algorithms show parallelism opportunities in other sections of their algorithmic designs, i.e., the application of evolutionary operators in NSGA-II and SPEA2, quality indicator computations in the indicator-based metaheuristics, the firefly movements in MO-FA, etc. Such operations should be also taken into consideration for parallelization purposes, in order to minimize the serial fraction of the application. Finally, data dependencies in these algorithms are mostly related to data structure management operations and must be ensured to avoid incorrect outcomes.

7.2 Shared-Memory Approaches

This section addresses the parallelization of our metaheuristics for phylogenetic inference on shared-memory multicore multiprocessor systems. In this kind of hardware setups, the system is composed of multiple processors, each one containing multiple cores, which share the same memory address space. This globally shared memory represents the element where all the processors coordination and synchronization take place. The development of parallel algorithms for shared-memory environments can be undertaken by using the OpenMP library [29]. An OpenMP application follows the fork-join model, where a main execution thread splits into multiple threads which perform tasks in parallel, resuming the serial execution upon completion of the parallel tasks. The fork/join blocks in OpenMP are known as parallel regions, which define a thread pool which can be used in accordance with the requirements of parallel processing of the algorithm. Those tasks showing parallelism opportunities can be distributed among the threads in the pool by using worksharing directives. OpenMP also provides directives to allow the definition of sequential sections of code to be executed by a single thread, atomic operations, synchronization mechanisms, etc. More details on OpenMP programming are available in Chapter 4 - Section 4.2.2 and reference [29].

In order to exploit pure shared-memory environments, two main parallelization schemes for our multiobjective metaheuristics are examined:

- Firstly, synchronous generational designs which parallelize the main sections that compose each generation of the algorithms, in such a way that the next generation does not take place until all the tasks in the previous one have been completed.
- Secondly, asynchronous non-generational approaches designed to allow each execution thread to perform parallel tasks continuously, modelling execution threads as asynchronous independent entities and dismissing the traditional concept of generation.

With the introduction of these two parallelization trends, we are interested in discussing which parallel design leads to an improved exploitation of the underlying hardware resources. In this context, we must also keep in mind the question on the outcomes quality, verifying if the fact of using one parallelization scheme or another makes an impact on the overall quality of the multiobjective results obtained by the metaheuristics.

7.2.1 Synchronous Generational Designs

The first proposal introduced to parallelize the studied metaheuristics consists of designing synchronous generational schemes. In such designs, all the parallel tasks which belong to each section of the algorithm must finish before the next section takes places. Moreover, a new generation can only be started after completing all the tasks in the current generation. In this way, the parallel version of the metaheuristic maintains the same properties and behaviour as its serial counterpart, while minimizing execution time by parallelizing the main sections which compose each iteration of the algorithm. Algorithm 14 shows an OpenMP-based pseudocode for this parallelization scheme.

Algorithm 14 Generic Synchronous Generational Parallel Scheme

```
1:  Initialize Data Structures (P, P', ParetoFront)
2:  #pragma omp parallel num_threads (num_threads)
3:  P ← Initialize Population (popSize, dataset, num_threads)
4:  while ! stop criterion reached (maxEvaluations) do
5:     #pragma omp single
6:        P ← Perform Fitness Assignments (P)
7:        P ← Perform Environmental Selections (P)
8:     #pragma omp for schedule (scheduleType)
9:     for i = 1 to popSize do
10:       P'_i.m ← Apply Evolutionary Operators (P)
11:       P'_i.τ ← Infer Phylogenetic Tree (P'_i.m, dataset)
12:       P'_i.scores ← Evaluate Solution (P'_i.τ, dataset)
13:    end for
14:    #pragma omp single
15:       ParetoFront ← Update Pareto Front (P, P', ParetoFront)
16: end while
```

This synchronous parallelization scheme proceeds as follows. At the beginning, a pool of *num_threads* OpenMP threads is created by using the *#pragma omp parallel* directive (line 2 in Algorithm 14), defining a parallel region which encloses the main loop of the metaheuristic. At each generation, parallelizable and non-parallelizable sections can be distinguished. For instance, the fitness assignments and environmental selections could potentially be affected by data dependencies issues, while also leading to possible read/write hazards as they make changes over the population data structure by using the information contained in that structure. These kinds of operations are then suitable to be enclosed in a single section, defined by the *#pragma omp single* directive (lines 5-7). On the other hand, the loops associated to the generation of new candidate distance matrices by means of evolutionary operators, the inference and optimization of the phylogenies associated to the computed matrices, and the evaluation of the resulting solutions is parallelized through the use of *#pragma omp for* worksharing directives (lines 8-13). It should be noted that these parallel loops might be affected by load imbalance issues, due to the fact that the operations contained in these loops could require different execution times (in accordance with the characteristics of the evolutionary operators and the topological structure of the inferred phylogenetic trees). In order to address this problem, dynamic scheduling policies can be considered in the worksharing directives. Finally, this pseudocode gives account of how the synchronous generational design imposes that all the tasks in the current generation must finish before starting a new one.

The OpenMP-based parallelization scheme in Algorithm 14 is based on the idea of initializing a parallel region at the beginning of the algorithm, using the available execution threads according to the parallelization opportunities shown by the operations considered in the metaheuristic. This design aims to avoid the additional thread creation/deletion overhead introduced when using *#pragma omp parallel for* directives inside the main loop of the metaheuristic. Such approach would require

a continuous initialization and deletion of the parallel region at each generation, which could potentially lead to a negative impact in performance. In addition, thread affinity policies are considered by enabling the *GOMP_CPU_AFFINITY* flag to make each particular thread run always on the same processing core. In this way, an efficient use of cache memories can be accomplished by avoiding cache re-fills when a thread is moved to a different processor.

In the following subsections, we provide specific details on the parallelization of the different metaheuristics studied in this Thesis: MOABC, MO-FA, NSGA-II, SPEA2, IMOBA, and IBEA.

MOABC

Algorithm 15 shows a synchronous parallel design for MOABC. In this metaheuristic, the main parallel opportunities are given by the computations performed at the employed exploitation step (lines 5-11 in Algorithm 15), the onlooker exploitation step (lines 14-20), and the scout exploration step (lines 30-34). To parallelize these three sections, *#pragma omp for* worksharing directives are applied over their enclosing loops. On the other hand, the interactions between employed and onlooker bees (defined by fast non-dominated sorts, crowding computations, and selection probabilities calculations) along with other data structure management operations at the end of the generation are performed by a single thread via *#pragma omp single* (lines 12-13, 21-29, and 35-36).

In this parallel implementation, load balance mechanisms are considered to parallelize the employed, onlooker, and scout bee loops. For the case of employed and onlooker bees, these mechanisms are implemented by introducing dynamic scheduling clauses, which assign new tasks to those threads which finish their initial workload. Regarding scout bees, the original design of the serial algorithm required to check, for each bee in the swarm, if its associated solution had reached the limit control parameter. Only if this condition was true, a scout search was performed. A straightforward parallelization of this loop would lead to load balance issues due to the if-condition. Furthermore, using dynamic scheduling over the entire swarm is not enough to solve the problem, as it could involve the assignment of null tasks to threads, giving rise to noticeable thread management times. Therefore, an efficient parallelization of this step requires conducting the loop only over the set of possible exhausted solutions. For this reason, we first compute how many scout searches must be carried out (lines 21-29), saving the number of exhausted solutions (*numExhSolutions*) along with their indices in an auxiliary array (*indexScoutBees*). Then, the scout bee loop is performed in a parallel way over the *numExhSolutions* bees that must take the scout role.

MO-FA

Algorithm 16 presents the parallel design of MO-FA for shared-memory environments, which aims to distribute the computations at the firefly movement loop among multiple execution threads (lines 16-22 in Algorithm 16). In order to implement this parallel version, two key issues must be considered. Firstly, as movements are applied over a variant number of dominated fireflies (if-condition) which learn from multiple fireflies in the swarm, load balancing techniques are required to make a proper

Algorithm 15 MOABC - Synchronous Parallel Pseudocode

```
 1: Initialize Data Structures (P, ParetoFront)
 2: #pragma omp parallel num_threads (num_threads)
 3: P ← Initialize Population (popSize/2, dataset, num_threads)
 4: while ! stop criterion reached (maxEvaluations) do
 5:     #pragma omp for schedule (scheduleType) /* Parallelizing the employed bees exploitation step */
 6:     for i = 1 to popSize/2 do
 7:         P'_i.m ← Generate Neighbour Employed Matrix (P_i.m, P, mutationRate)
 8:         P'_i.τ ← Infer Phylogenetic Tree (P'_i.m, dataset)
 9:         P'_i.scores ← Evaluate Solution (P'_i.τ, dataset)
10:         P_i ← Update Employed Solution (P_i, P'_i)
11:     end for
12:     #pragma omp single
13:         P, probabilityVector ← Perform Employed-Onlooker Interactions (P, popSize/2)
14:     #pragma omp for schedule (scheduleType) /* Parallelizing the onlooker bees exploitation step */
15:     for i = (popSize/2)+1 to popSize do
16:         P_i, P'_i.m ← Select and Generate Neighbour Onlooker Matrix (P, mutationRate, probabilityVector)
17:         P'_i.τ ← Infer Phylogenetic Tree (P'_i.m, dataset)
18:         P'_i.scores ← Evaluate Solution (P'_i.τ, dataset)
19:         P_i ← Update Onlooker Solution (P_i, P'_i)
20:     end for
21:     /* Computing the number of exhausted solutions for the scout bees step */
22:     #pragma omp single
23:         numExhSolutions ← 0
24:         for i = 1 to popSize do
25:             if P_i.counter > limit then
26:                 indexScoutBees[numExhSolutions] ← i
27:                 numExhSolutions ← numExhSolutions + 1
28:             end if
29:         end for
30:     #pragma omp for schedule (scheduleType) /* Parallelizing the scout bees exploration step */
31:     for i = 1 to numExhSolutions do
32:         P_indexScoutBees[i] ← Generate Scout Bee (P_indexScoutBees[i], mutationRate)
33:         P_indexScoutBees[i].scores ← Evaluate Solution (P_indexScoutBees[i].τ, dataset)
34:     end for
35:     #pragma omp single
36:         Perform Final Steps of the Generation(P, ParetoFront)
37: end while
```

distribution of tasks among threads. In this way, the algorithm's performance can be improved by avoiding idle threads throughout the execution of the metaheuristic. Secondly, as multiple threads operate simultaneously over the swarm, additional supporting data structures must be introduced to handle possible read-write hazards in the movement computations.

These mechanisms are implemented in the following way. In order to minimize load balance issues, the number of currently dominated fireflies in the swarm is calculated at the beginning of each generation (lines 5-14). By using this strategy, the movement loop will consider *numDominatedFireflies* update tasks to be processed by the execution threads via worksharing directives, instead of the *popSize* iterations defined in the original serial design (which could potentially involve null tasks and, therefore, lead to performance issues). In addition, as dominated fireflies must learn from a changing number of fireflies in accordance with the dominance concept, OpenMP dynamic scheduling techniques are applied to distribute new tasks among those threads that have completed their initial workload. Finally, in order to address read-write hazards, a backup population *auxPop* is introduced (line 15). This auxiliary population is defined to avoid inconsistencies when updating fireflies, keeping a stable copy of the current state of the swarm.

Algorithm 16 MO-FA - Synchronous Parallel Pseudocode

```
1:  Initialize Data Structures (P, ParetoFront)
2:  #pragma omp parallel num_threads (num_threads)
3:  P ← Initialize Population (popSize, dataset, num_threads)
4:  while ! stop criterion reached (maxEvaluations) do
5:      #pragma omp single
6:          /* Compute the number of dominated fireflies */
7:          idDominatedFireflies ← 0
8:          numDominatedFireflies ← 0
9:          for i = 1 to popSize do
10:             if ∃ P_j: P_j ≻ P_i then
11:                 idDominatedFireflies[numDominatedFireflies] ← i
12:                 numDominatedFireflies ← numDominatedFireflies + 1
13:             end if
14:         end for
15:         auxPop ← P /* Saving the current state of the swarm */
16:     #pragma omp for schedule (scheduleType) /* Parallelizing firefly movement loop */
17:     for i = 1 to numDominatedFireflies do
18:         idDom ← idDominatedFireflies[i]
19:         P_idDom.m ← Attract Firefly (P_idDom.m,auxPop,β0,γ,α)
20:         P_idDom.τ ← Infer Phylogenetic Tree (P_idDom.m, dataset)
21:         P_idDom.scores ← Evaluate Solution (P_idDom.τ, dataset)
22:     end for
23:     #pragma omp single
24:         Perform Final Steps of the Generation(P, ParetoFront)
25: end while
```

NSGA-II

An OpenMP-based design to parallelize NSGA-II is shown in Algorithm 17. In this algorithm, the major source of parallelism is given by the generation of the offspring population at each generation. More specifically, the *popSize* iterations of the offspring loop can be distributed among multiple execution threads, using for this purpose worksharing directives (lines 9-14 in Algorithm 17). This parallel loop can benefit from dynamic scheduling policies, due to the fact that different iterations may require different processing times, according to the stochastic nature of the evolutionary operators and the topological features of the phylogenetic tree to be inferred and evaluated. The remaining operations in NSGA-II involve the management of the population structures handled by the algorithm (fast non-dominated sorts, crowding distance computations, parent population update, etc.), being enclosed by *#pragma omp single* directives (lines 5-8 and 15-16).

SPEA2

Algorithm 18 introduces the pseudocode for the shared-memory parallel version of SPEA2. This parallel design resembles the parallelization scheme applied in NSGA-II, as both algorithms show their main parallelism opportunities at the offspring generation loop. Under this perspective, the OpenMP-based approach introduces *#pragma omp for* directives to parallelize the generation and evaluation of new candidate solutions (lines 9-14 in Algorithm 18). Dynamic scheduling mechanisms are configured to reduce load imbalances due to the potential existence of divergent times in the computation of evolutionary operators and the inference and evaluation of topologies. On the other hand, those operations which involve data dependencies and data structures management (including

Algorithm 17 NSGA-II - Synchronous Parallel Pseudocode

```
1:  Initialize Data Structures (C, Q, P, ParetoFront)
2:  #pragma omp parallel num_threads (num_threads)
3:  P ← Initialize Population (popSize, dataset, num_threads)
4:  while ! stop criterion reached (maxEvaluations) do
5:      #pragma omp single
6:          C ← P ∪ Q
7:          Fronts ← Fast Non-Dominated Sort and Crowding Computation (C)
8:          P ← Update Parent Population (Fronts)
9:      #pragma omp for schedule (scheduleType) /* Parallelizing the offspring generation and evaluation */
10:     for i = 1 to popSize do
11:         Q_i.m ← Apply Evolutionary Operators (P, crossoverProb, mutationProb)
12:         Q_i.τ ← Infer Phylogenetic Tree (Q_i.m, dataset)
13:         Q_i.scores ← Evaluate Solution (Q_i.τ, dataset)
14:     end for
15:     #pragma omp single
16:         Perform Final Steps of the Generation(P, Q, ParetoFront)
17: end while
```

Algorithm 18 SPEA2 - Synchronous Parallel Pseudocode

```
1:  Initialize Data Structures (P, P̄, ParetoFront)
2:  #pragma omp parallel num_threads (num_threads)
3:  P ← Initialize Population (popSize, dataset, num_threads)
4:  while ! stop criterion reached (maxEvaluations) do
5:      #pragma omp single
6:          P,P̄ ← Assign Fitness (P,P̄)
7:          multiSet ← Fitness-Based Sorting (P ∪ P̄)
8:          P̄ ← Perform Environmental Selection (multiSet)
9:      #pragma omp for schedule (scheduleType) /* Parallelizing the offspring generation and evaluation */
10:     for i = 1 to popSize do
11:         P_i.m ← Apply Evolutionary Operators (P̄, crossoverProb, mutationProb)
12:         P_i.τ ← Infer Phylogenetic Tree (P_i.m, dataset)
13:         P_i.scores ← Evaluate Solution (P_i.τ, dataset)
14:     end for
15:     #pragma omp single
16:         Perform Final Steps of the Generation(P, P̄, ParetoFront)
17: end while
```

the SPEA2 fitness assignment step and the archive update in the environmental selection procedure) are carried out by applying *#pragma omp single* directives (lines 5-8 and 15-16).

IMOBA

Regarding the parallelization of IMOBA, its synchronous parallel design is reported in Algorithm 19. In an indicator-based design, we can take advantage of the fact that the computation of I_{HD} values for a certain individual show no dependencies with regard to other ones, with the aim of minimizing the times introduced when considering quality indicators for fitness evaluation purposes. Following this assumption, the parallelization of indicator calculations is included in this parallel proposal (as suggested in lines 5 and 15 in Algorithm 19).

In addition, *#pragma omp for* directives are applied over the most time-consuming loops of the algorithm, which are related to the two main sections of IMOBA: the bat movement step (lines 9-14) and the bat search step (lines 20-29). It must be considered that the bat search step is governed by an if-else condition that decides if a thread executes an exploitation procedure or an exploration one. As

Algorithm 19 IMOBA - Synchronous Parallel Pseudocode

```
1:  Initialize Data Structures (P, P', ParetoFront)
2:  #pragma omp parallel num_threads (num_threads)
3:  P ← Initialize Population (popSize, dataset, f_max, f_min, A_0, r_max, num_threads)
4:  while ! stop criterion reached (maxEvaluations) do
5:      indicatorValues ← Calculate Indicator Values (P, Z_ref, num_threads) /* indicator computations in parallel */
6:      #pragma omp single
7:          P ← Assign Indicator-Based Fitness (indicatorValues, κ)
8:          P ← Environmental Selection (P, κ)
9:      #pragma omp for schedule (scheduleType) /* Parallelizing bat movements */
10:     for i = 1 to popSize do
11:         P'_i.m ← Move Bat (P_i, f_max, f_min)
12:         P'_i.τ ← Infer Phylogenetic Tree (P'_i.m, dataset)
13:         P'_i.scores ← Evaluate Solution (P'_i.τ, dataset)
14:     end for
15:     indicatorValues ← Calculate Indicator Values (P ∪ P', Z_ref, num_threads) /* indicator computations in parallel */
16:     #pragma omp single
17:         P ∪ P' ← Assign Indicator-Based Fitness (indicatorValues, κ)
18:         P ← Environmental Selection (P ∪ P', κ)
19:         selectionArray ← Compute Selection Probabilities (P, popSize)
20:     #pragma omp for schedule (scheduleType) /* Parallelizing bat searches */
21:     for i = 1 to popSize do
22:         if rand > P_i.r then
23:             P'_i.m ← Generate Exploitation Bat (selectionArray, mutationProb)
24:         else
25:             P'_i.m ← Generate Exploration Bat (mutationProb)
26:         end if
27:         P'_i.τ ← Infer Phylogenetic Tree (P'_i.m, dataset)
28:         P'_i.scores ← Evaluate Solution (P'_i.τ, dataset)
29:     end for
30:     #pragma omp single
31:         Perform Final Steps of the Generation(P ∪ P', ParetoFront, α, γ)
32: end while
```

a result, this loop can be a potential source of load imbalance from a parallel processing perspective. In order to improve load balance in this main section of the algorithm, we rely on scheduling policies to enable a dynamic assignment of iterations among the threads involved in the loop computations. Finally, fitness assignments, environmental selections, the computation of the selection array, and other management operations are enclosed by *single* directives (lines 6-8, 16-19, and 30-31).

IBEA

The design proposed to parallelize IBEA on shared-memory systems is provided in Algorithm 20. This metaheuristic shows two main sections where parallelism can be applied: the computation of quality indicator values and the generation of offspring solutions by means of evolutionary operators. Firstly, as suggested in IMOBA, the loops for computing I_{HD} values are suitable to be parallelized, using for this purpose a *#pragma omp for* directive (lines 5-11 in Algorithm 20). Please observe that this structure is also the one applied to integrate parallelism into indicator calculations in IMOBA (lines 5 and 15 in Algorithm 19). Secondly, as in NSGA-II and SPEA2, the generation and evaluation of the offspring population is parallelized via worksharing with dynamic scheduling policies (lines 15-20). Finally, *#pragma omp single* directives are used to carry out data-dependent functions and data structure management operations (lines 12-14 and 21-22).

Algorithm 20 IBEA - Synchronous Parallel Pseudocode

```
1:  Initialize Data Structures (P, P', ParetoFront)
2:  #pragma omp parallel num_threads (num_threads)
3:  P ← Initialize Population (popSize, dataset, num_threads)
4:  while ! stop criterion reached (maxEvaluations) do
5:      /* Parallelizing the computation of indicator values */
6:      #pragma omp for schedule (scheduleType)
7:      for i = 1 to |P| do
8:          for j = 1 to |P| do
9:              indicatorValues[i][j] ← Calculate I_HD Values (P_i, P_j, Z_ref)
10:         end for
11:     end for
12:     #pragma omp single
13:         P ← Assign Indicator-Based Fitness (indicatorValues, κ)
14:         P ← Environmental Selection (P, κ)
15:     #pragma omp for schedule (scheduleType) /* Parallelizing the offspring generation and evaluation */
16:     for i = 1 to popSize do
17:         P'_i.m ← Apply Evolutionary Operators (P, crossoverProb, mutationProb)
18:         P'_i.τ ← Infer Phylogenetic Tree (P'_i.m, dataset)
19:         P'_i.scores ← Evaluate Solution (P'_i.τ, dataset)
20:     end for
21:     #pragma omp single
22:         Perform Final Steps of the Generation(P∪P', ParetoFront)
23: end while
```

7.2.2 Asynchronous Non-Generational Designs

In recent years, the research on parallel metaheuristics has turned its interest to the study of asynchronous parallelization schemes [92], which are defined to maximize the exploitation of parallel architectures by modelling execution threads as asynchronous independent entities. These new parallel designs represent a promising way to address the computational costs required to solve MOPs. The main research efforts to apply asynchronous strategies to multiobjective algorithms have being focused on parallelizing the well-known NSGA-II [52]. In the literature, we can find NSGA-II implementations which integrate asynchronous intra-algorithm parallelization to solve optimization problems with heterogeneous evaluation costs [189], real world applications on distributed architectures [54], and time-constrained problems [199]. Furthermore, island-based models with asynchronous migration were also proposed, leading to promising results in time-demanding applications [113].

As example of the application of this kind of parallelization schemes, we study the implementation of an asynchronous non-generational design for MOABC, named as AN-MOABC. This metaheuristic was chosen due to the fact that honey bees in nature represent true asynchronous agents and, therefore, a nature-inspired asynchronous parallelization model for this algorithm can be modelled on the basis of the real behaviour of these social insects. More specifically, the synchronous implementation reported in the previous section (which will be denoted as SG-MOABC) is based on the assumption that, when a bee finishes its tasks, it must wait for the remaining bees in order to carry out new tasks. Actually, honey bees are not governed by such synchronous rules. When a bee has finished the exploitation/exploration of a food source, it returns immediately to the hive without waiting for the remaining bees. Furthermore, a bee may leave the hive to carry out new tasks without waiting for the arrival or leaving of other bees. Hence, the main goal of AN-MOABC is to define an

asynchronous design to parallelize MOABC, discarding the traditional concept of generation, with the idea of maximizing the exploitation of hardware resources on shared-memory environments.

An Asynchronous Design for MOABC - General Features

In order to model the asynchronous behaviour of honey bees, AN-MOABC follows a master-worker scheme, as it represents a suitable design to parallelize multiobjective bioinspired algorithms [177]. Our model organizes parallel execution threads under the following roles:

1. **Master**. In this design, the master thread will represent the hive itself. It will manage all the shared data structures (swarms/populations and probability vectors) and store the solutions found by worker threads as they are generated.

2. **Workers**. These threads will carry out parallel tasks involving the exploitation/exploration of solutions. After completing a task, a thread with the worker role communicates the results to the master and begins a new task without waiting for other worker threads. Two kinds of workers are distinguished: employed workers (defined to perform employed bee tasks) and onlooker workers (to carry out onlooker bee tasks).

Algorithm 21 shows the OpenMP-based parallelization scheme in AN-MOABC. This algorithm defines a parallel region in which execution threads assume the different roles identified previously, using thread affinity policies in the same way as in the synchronous counterpart. The master role is assigned to the thread with identifier *#num_threads-1*, defining the first *num_threads/2* threads as employed workers, and the remaining *num_threads/2-1* as onlooker workers.

Algorithm 21 AN-MOABC Parallelization Scheme

```
1:  /* Initializing communication/interaction structures */
2:  BeeFIFO ← new std::queue<Bee*> [num_threads-1]
3:  FIFOSem ← new Semaphore [num_threads-1]
4:  SwarmSem ← new Semaphore [num_threads-1]
5:  #pragma omp parallel num_threads (num_threads)
6:  id ← omp_get_thread_num()
7:  if id == num_threads-1 then
8:      Do Master Tasks (BeeFIFO, FIFOSem, SwarmSem, ConsSwarm, InconsSwarm)
9:  else
10:     if id < num_threads/2 then
11:         Do Employed Worker Tasks (BeeFIFO[id], FIFOSem[id], SwarmSem[id], ConsSwarm)
12:     else
13:         Do Onlooker Worker Tasks (BeeFIFO[id], FIFOSem[id], SwarmSem[id], ConsSwarm)
14:     end if
15: end if
```

Task assignments and interactions with the master thread are modelled by defining different data structures. Firstly, AN-MOABC distinguishes between *consistent* and *inconsistent* swarms. The current state of the population and the selection probability vector is contained in the consistent swarm (*ConsSwarm* in Algorithm 21), which is accessed by worker threads to read the bee which contains the starting point of the task. The results of worker tasks will be stored in the inconsistent swarm (*InconsSwarm*) which is only accessed by the master thread. In this way, the inconsistent

swarm is dynamically updated by the master with the solutions found by the workers, computing new selection probabilities for these solutions. In order to make the population evolve, the consistent swarm will be updated by the master thread with the information stored in the inconsistent swarm.

The communication of results from the workers to the master is conducted by defining queue data structures (*BeeFIFO*). Each worker will store the resulting solutions into a particular queue, which will be read by the master to update the inconsistent swarm. By defining a queue per worker, we aim to avoid write hazards due to multiple threads inserting multiple solutions concurrently in a single globally shared queue. In addition, we need to provide mechanisms to ensure data integrity when master and worker threads perform read/write operations over the same data structure. For this purpose, binary semaphores are defined to guarantee master-worker mutual exclusion in the queues (*FIFOSem*) and the consistent swarm (*SwarmSem*).

Master Tasks

Master tasks are described in Algorithm 22. Initially, the master thread remains waiting for the arrival of new solutions from the workers, checking the contents of each queue *BeeFIFO* (lines 3-10 in Algorithm 22). Once new solutions have been detected, the master proceeds to store them, updating the state of the inconsistent swarm. Over the updated inconsistent swarm, the master carries out a fast non-dominated sort and calculates crowding distances, computing afterwards the new array of selection probabilities for onlooker workers (lines 12-13).

Algorithm 22 AN-MOABC - Master Thread Tasks

```
 1: while ! stop criterion reached (maxEvaluations) do
 2:     update ← false
 3:     for i = 0 to num_threads-2 do
 4:         while BeeFIFO[i] has elements do
 5:             FIFOSem[i].lock()
 6:             InconsSwarm.P_beeId ← BeeFIFO[i].pop()
 7:             FIFOSem[i].unlock()
 8:             update ← true
 9:         end while
10:     end for
11:     if update == true then
12:         InconsSwarm.P ← Fast Non Dominated Sort and Crowding Computation (InconsSwarm.P, popSize)
13:         InconsSwarm.probabilityVector ← Compute Selection Array (InconsSwarm.P, popSize/2)
14:         SwarmSem[i].lock() /* ∀i : i = 0 to num_threads − 2 */
15:         Interchange Swarm Structures Pointers (ConsSwarm, InconsSwarm)
16:         SwarmSem[i].unlock() /* ∀i : i = 0 to num_threads − 2 */
17:         ParetoFront ← Update Pareto Front (ConsSwarm.P, ParetoFront)
18:     end if
19: end while
20: Send Termination Signal (i) /* ∀i : i = 0 to num_threads − 2 */
```

As worker threads only have access to a swarm containing outdated solutions at that moment, the new status in the inconsistent swarm must be communicated to the consistent structure. For this purpose, the master carries out the transmission of information from the inconsistent swarm to the consistent one through the interchange of their pointers (lines 14-16). This step is enclosed in a critical section by using lock/unlock semaphore primitives, in order to guarantee that no other thread

is reading the consistent swarm at the moment this structure is updated. After that, the new status of the population will be available to worker threads. In addition, the set of Pareto solutions is updated with the solutions included in the new consistent swarm (line 17).

These actions are repeated until the stop criterion is reached. This situation is notified to the workers by sending termination signals (line 20).

Worker Tasks

Worker tasks involve the reading/selection of previously found solutions stored in the consistent swarm, the computation of new candidate matrices, and the reconstruction and evaluation of the corresponding phylogenetic trees. These tasks are performed attending to each worker role: employed workers carry out their tasks according to the employed bees step, while onlooker workers follow onlooker bees guidelines. Algorithm 23 provides the pseudocode for the worker threads.

Algorithm 23 AN-MOABC - Worker Thread Tasks

```
1:  beeIds ← Compute Bee Indices Per Thread (popSize, num_threads, threadID)
2:  while ! Termination Signal received do
3:      SwarmSem[threadID].lock()
4:      if threadID < num_threads/2 then
5:          beeSolution, neighbourSolution ← Get Employed Bee (ConsSwarm.P, beeIds)
6:      else
7:          beeSolution ← Get Onlooker Bee (ConsSwarm.P, ConsSwarm.probabilityVector, beeIds)
8:      end if
9:      SwarmSem[threadID].unlock()
10:     if threadID < num_threads/2 then
11:         Perform Employed Bee Tasks (beeSolution, neighbourSolution, mutationRate, dataset)
12:     else
13:         Perform Onlooker Bee Tasks (beeSolution, mutationRate, dataset)
14:     end if
15:     if beeSolution.counter > limit then
16:         Perform Scout Bee Tasks (beeSolution, mutationRate, dataset)
17:     end if
18:     FIFOSem[threadID].lock()
19:     BeeFIFO[threadID].push(beeSolution)
20:     FIFOSem[threadID].unlock()
21: end while
```

As the size of the population (parameter *popSize*) may not match the number of workers, these threads firstly compute the positions in the consistent swarm that will be processed by each one (line 1 in Algorithm 23). Hence, *popSize/(num_threads-1)* bees in the consistent swarm will be assigned to each worker, where the first *popSize/2* bees must be processed by employed workers and the remaining ones by onlooker workers.

Worker actions start with the reading/selection of a solution maintained in the consistent swarm (lines 3-9), in accordance with the role taken by the worker (employed or onlooker). Afterwards, a new solution is generated by performing employed/onlooker bee exploitations (lines 10-14). If an exhausted solution is detected, the worker will carry out scout bee searches to generate a new solution which replaces the abandoned one (lines 15-17). Finally, the obtained results are communicated to the master thread via queues (lines 18-20). This loop is repeated without waiting for any other worker thread (non-generational approach), until the termination signal is received from the master.

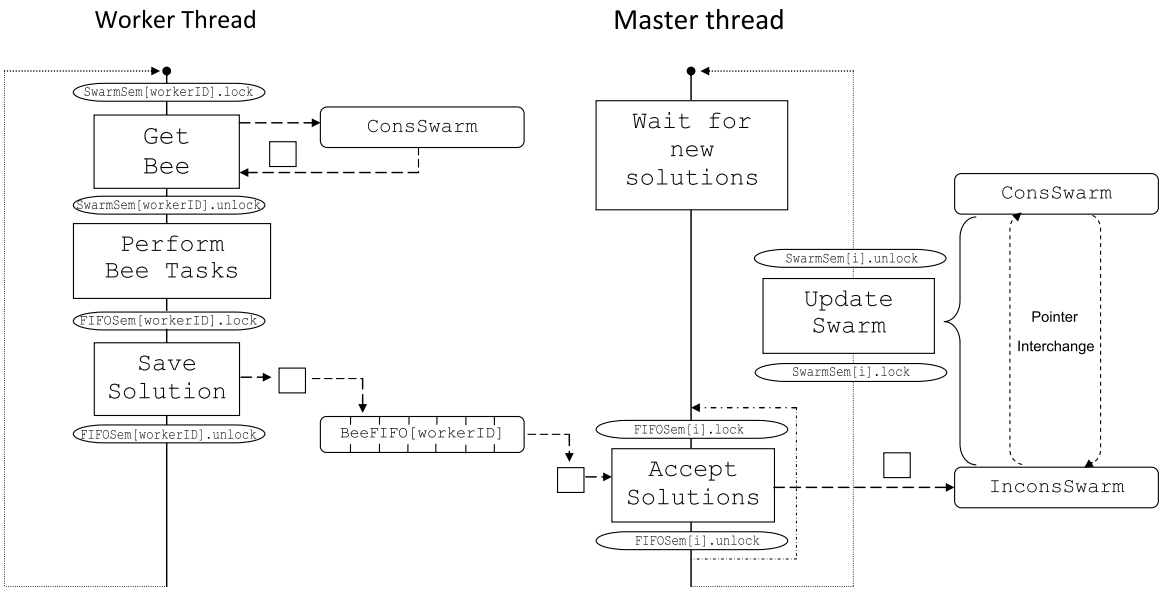


Fig. 7.2 Master-worker interactions in the asynchronous parallel implementation of MOABC

Master-worker shared structures are used in the reading/selection and the results communication steps. Firstly, worker threads must read the consistent swarm in order to obtain the solutions to be exploited. This task must be done inside a critical section (defined by the *SwarmSem* semaphores) to avoid situations where the master updates the consistent swarm pointer while worker threads are reading it. It should be noted that multiple workers can read the consistent swarm simultaneously, as the critical section only applies over concurrent master-worker conflicting accesses. Secondly, data integrity mechanisms are also applied when storing results in the queues, using the *FIFOSem* semaphores. Figure 7.2 shows an example of master-worker interactions in this parallel design. In the next chapter, a thorough comparative analysis of parallel performance between this asynchronous design and the synchronous one will be undertaken to verify the relevance of the proposal.

7.3 Mixed Mode Shared-Distributed Memory Approaches

Current cluster platforms are usually composed of a set of computing nodes linked by an interconnection network where each node contains a multicore (multi)processor. This configuration defines a hybrid shared-distributed memory organization in which multicore programming and message passing techniques are combined to exploit the strength of both models [53]. In this way, we can take advantage of the scalability of distributed-memory systems and the efficiency, memory savings, and easy programming of shared-memory systems. While other computing devices can also be integrated into the platform (i.e. GPUs) in a heterogeneous configuration, our focus here is on the development of parallel designs for clustered multicore multiprocessors.

One of the most popular paradigms for programming this kind of systems is the mixed mode model based on the standards OpenMP and MPI [4]. A mixed mode (or hybrid) parallel design

defines a set of MPI processes which are located at the different nodes contained in the cluster architecture. Each process is composed of several OpenMP execution threads defined to carry out an intra-node shared-memory parallelization of the assigned tasks, which are mapped to their local memories via MPI messages. This paradigm allows us to combine shared-memory and distributed-memory properties to take advantage of the underlying hardware resources. More details on MPI+OpenMP hybrid programming can be found in Chapter 4 - Section 4.2.5 and reference [74].

In this section, we study the parallelization of multiobjective metaheuristics on multicore clusters by using mixed mode programming. To this end, three metaheuristics with different algorithmic features have been chosen in order to analyze their parallel performance on this kind of systems: MOFA (representing a swarm intelligence design with dynamic workload), NSGA-II (dominance-based evolutionary design with static workload), and IBEA (indicator-based design with static workload).

7.3.1 Hybrid Parallel Design

Algorithm 24 provides a generic hybrid design for the parallelization of our metaheuristics for phylogenetics under a master-worker scheme. This mixed mode approach defines a set of *nproc* MPI processes located at different computing nodes, introducing intra-node parallelization by defining *num_threads* OpenMP threads per process. After initializing the MPI process, a *#pragma omp parallel* directive is included to define an OpenMP parallel region which encloses the code associated to each process (lines 1 and 2 in Algorithm 24). The idea is to take advantage of OpenMP features to conduct a multithreaded processing of the tasks assigned to the MPI processes, in accordance with the parallelism opportunities shown by the metaheuristic. In order to guarantee thread safety, MPI processes are initialized by using an *MPI_THREAD_SERIALIZED* configuration which allows any thread to call MPI primitives inside a *#pragma omp single* region.

In the proposed design, one of the processes (with identifier *id=0*) assumes the role of master and performs algorithmic-specific operations over a population of *popSize* individuals to generate a set of candidate solutions in the shape of distance matrices (line 5). These matrices are distributed for processing purposes among the remaining processes, which take the role of workers (with identifiers *id=1* to *nproc-1*). Workers tasks (as described in Algorithm 25) involve the reconstruction and evaluation of phylogenetic topologies from the assigned distance matrices, which represent the most time-consuming operations identified in the application profile analysis (Section 7.1.1). The assignment of *tasksNumber/nproc* tasks (matrices) per process is carried out by using MPI communication primitives (lines 7 and 10). In addition, the pseudocode in Algorithm 24 suggests that the master process changes its role to worker, a strategy adopted to make this process contribute in worker computations with the aim of avoiding idle resources. After processing the assigned tasks (line 13), the results obtained by the workers (given by the inferred phylogenetic trees and their objective function scores) are sent to the master process via MPI messages (lines 16 and 20). These steps are repeated until the stop criterion of the metaheuristic is reached, finishing the execution of the processes involved in the computations (lines 23-24).

Algorithm 24 Generic Hybrid Parallelization Scheme

```
1:  MPI_Initialize (MPI_THREAD_SERIALIZED)
2:  #pragma omp parallel num_threads (num_threads)
3:  while ! stop criterion reached (maxEvaluations) do
4:    if master process /* id = 0 */ then
5:      Perform Master Tasks (P, num_threads)
6:      #pragma omp single
7:        MPI_Send (P.m, tasksNumber/nproc, i) /* ∀ i: i=1 to nproc-1 */
8:    else
9:      #pragma omp single
10:       MPI_Receive (P.m, tasksNumber/nproc, master)
11:   end if
12:   /* ∀ i: i=0 to nproc-1 */
13:   Perform Worker Tasks Using Multithreading Features (P, num_threads)
14:   if master process then
15:     #pragma omp single
16:       MPI_Receive (Results, tasksNumber/nproc, i) /* ∀ i: i=1 to nproc-1 */
17:       ParetoFront ← Update Pareto Front (Results, ParetoFront)
18:   else
19:     #pragma omp single
20:       MPI_Send (Results, tasksNumber/nproc, master)
21:   end if
22: end while
23: Send/Receive Termination Signals (nproc)
24: MPI_Finalize ()
```

Algorithm 25 Worker Tasks for the Hybrid Parallelization Scheme

```
1: #pragma omp for schedule (scheduleType)
2: for j = 1 to tasksNumber/nproc do
3:   P_j.τ ← Infer Phylogenetic Tree (P_j.m, dataset)
4:   P_j.scores ← Evaluate Solution (P_j.τ, dataset)
5: end for
```

Communications and Synchronizations

Two important factors which determine the success of mixed mode parallel designs are the communication and synchronization patterns. Communications and synchronizations represent potential overheads with a strong influence on parallel performance, so the development of hybrid parallel approaches must introduce proper mechanisms to reduce the impact of these two factors on scalability.

Regarding communication patterns, message passing takes place in the task assignment and results delivery steps. In order to conduct worker computations, the communication of *tasksNumber/nproc NxN* floating-point matrices per process is required (where N is the number of species in the input alignment). As the message delivery time depends on the size and datatype of the communicated structures, the communication overhead could rise significantly when increasing values of N

are considered. With the aim of reducing such times, our hybrid designs take advantage of the symmetry shown by distance matrices to define messages which comprise distance arrays containing the matrix entries located below the main diagonal. In the results communication step, this same strategy is used to communicate the optimized matrices, along with the generated topologies and objective function scores. In addition, the master will receive results from workers progressively by using the flag MPI_ANY_SOURCE in MPI_Recv, achieving a partial communication/computation overlap.

An alternative to this implementation consists of sending Newick strings [59] instead of distance matrices. The Newick tree format allows the representation of a phylogenetic topology as a string of characters, requiring fewer bytes than the distance array approach. However, this alternative needs to infer previously the phylogenetic topology, that is, to carry out the BIONJ tree-building procedure exclusively in the master process. As BIONJ represents one of the operations that contribute most to the overall execution time, the use of Newick codes for communication purposes is not suitable to achieve an improved exploitation of cluster resources. Therefore, Newick strings are only considered in our communications to send the generated phylogenetic trees in the results delivery step.

Synchronization patterns must be analyzed at the OpenMP thread and MPI process levels. Thread synchronizations take place at implicit barriers introduced by worksharing directives, in which OpenMP threads wait for others to complete their tasks. In this sense, load balance issues may arise when parallelizing loops whose iterations involve variable times, leading to high thread synchronization times. To address this problem, we can make use of OpenMP loop scheduling policies. Secondly, inter-process synchronizations are given at the message-passing steps: task assignments and results communications. This upper-level of synchronizations depends on the differences in execution time shown by workers when processing computationally expensive tasks and also the ability of the parallel metaheuristic to define accurate workload distributions among processes and threads.

7.3.2 Implementation Details

On the basis of the parallelization scheme introduced in Algorithm 24, some details on the mixed mode implementations of MO-FA, NSGA-II, and SPEA2 are described next. In these implementations, master tasks will be defined according to the algorithmic design of each metaheuristic. The number of tasks to be assigned to the worker processes will also depend on the metaheuristic features, distinguishing between implementations with dynamic workload and static workload.

MO-FA

Algorithm 26 provides the implementation of master tasks in MO-FA, which mainly involve the movement of dominated fireflies. As observed in the implementation for shared-memory environments, a straightforward approach to parallelize the movement loop could give rise to load balance problems due to the if-condition which checks for dominated fireflies. In order to solve this issue, we apply the same strategy in the mixed mode implementation, verifying how many fireflies are dominated and saving their identifiers inside an array *idDominatedFireflies*. In this way, OpenMP threads

Algorithm 26 MO-FA - Master Tasks for the Mixed Mode Implementation

```
1:  #pragma omp single
2:      /* Computing the number of dominated fireflies */
3:      idDominatedFireflies ← Compute Dominated Fireflies (P)
4:      numDominatedFireflies ← idDominatedFireflies.size
5:      auxPop ← P /* Saving the current state of the swarm */
6:  #pragma omp for schedule (scheduleType) /* MO-FA movement loop */
7:  for j = 1 to numDominatedFireflies do
8:      idDom ← idDominatedFireflies[j]
9:      P_idDom.m ← Attract Firefly (P_idDom.m, auxPop, β0, γ, α)
10: end for
```

iterate over *numDominatedFireflies* individuals. Following this line, a backup population *auxPop* that contains the current state of the swarm is also used to avoid read/write hazards when updating distance matrices, processing afterwards with the parallel computation of firefly movements. By exploiting the parallel opportunities shown by the movement loop at the master process with OpenMP, we can reduce the times needed to prepare new tasks for the workers. It should be considered that the algorithmic design of MO-FA defines a dynamic number of worker tasks at each generation, given by the amount of dominated fireflies which have been detected. Therefore, the generated *numDominatedFireflies* candidate solutions are distributed among worker processes, which perform the inferences and evaluations defined as worker tasks (Algorithm 25) over their assigned workload.

NSGA-II

Master tasks in NSGA-II, as shown in Algorithm 27, consider firstly the management of the populations that must be carried out before generating new offspring solutions. By using the directive *#pragma omp single*, a single thread creates the combined population, performs the fast non-dominated sort and crowding distance computation, and updates the parent population. Afterwards, as the computation of evolutionary operators for different offspring individuals can be carried out concurrently, the loop for generating offspring matrices is parallelized by using *#pragma omp for*. After completing these operations, the master communicates the resulting *popSize* matrices in the offspring population Q to the workers. In this case, we are dealing with a fixed number of tasks to be processed by the workers, which is given by the size of Q (that is, the value defined by the *popSize* input parameter). This static workload is distributed among worker processes upon termination of master operations, proceeding with the inference and evaluation of the assigned offspring solutions.

IBEA

Regarding the mixed mode parallel implementation of IBEA, the definition of master tasks is available in Algorithm 28. At each generation, the master carries out the assessment of the population under the quality indicator, computing I_{HD} values, fitness scores for each individual in the population, and performing the environmental selection procedure. With the aim of minimizing I_{HD} times, the indicator computation loop is parallelized by using *#pragma omp for*. On the other hand, the fitness assignment and environmental selection steps are performed by a single thread. After carrying out

Algorithm 27 NSGA-II - Master Tasks for the Mixed Mode Implementation

```
1: #pragma omp single
2:     /* Population management */
3:     C ← P ∪ Q
4:     Fronts ← Fast Non-Dominated Sort and Crowding Computation (C)
5:     P ← Update Parent Population (Fronts)
6: #pragma omp for schedule (scheduleType) /* Offspring generation loop */
7: for j = 1 to popSize do
8:     Q_j.m ← Apply Evolutionary Operators (P, crossoverProb, mutationProb)
9: end for
```

Algorithm 28 IBEA - Master Tasks for the Mixed Mode Implementation

```
1: /* Parallelizing the computation of indicator values */
2: #pragma omp for schedule (scheduleType)
3: for j = 1 to |P| do
4:     for k = 1 to |P| do
5:         indicatorValues[j][k] ← Calculate I_HD Values (P_j, P_k, Z_ref)
6:     end for
7: end for
8: #pragma omp single
9:     /* Population management */
10:    P ← Assign Indicator-Based Fitness (indicatorValues, κ)
11:    P ← Environmental Selection (P, κ)
12: #pragma omp for schedule (scheduleType) /* Offspring generation loop */
13: for j = 1 to popSize do
14:    P'_j.m ← Apply Evolutionary Operators (P, crossoverProb, mutationProb)
15: end for
```

these operations, the master proceeds with the execution of evolutionary operators to generate new matrices in the offspring loop, which is also parallelized at the intra-process level by using OpenMP worksharing directives. Once the offspring matrices have been generated, the master is ready to assign tree inference and evaluation tasks to the worker processes. As in NSGA-II, a static number of *popSize* candidate solutions will be distributed for worker processing purposes.

The next chapter will address the experimental evaluation of these different hybrid algorithmic designs, verifying how accurately each metaheuristic takes advantage of the computing capabilities of cluster systems. For this purpose, different scenarios will be considered. Firstly, NSGA-II and MO-FA will be used to compare implementations with static/dynamic workload on a cluster configuration composed of six computing nodes (each one containing two quad-core processors). Secondly, the performance assessment of IBEA will be carried out over a constellation configuration comprising two computing nodes (each one composed of 24 processing cores), studying which configuration of MPI processes/OpenMP threads leads to the best exploitation of this kind of systems.

7.4 Summary

This chapter has undertaken the intra-algorithm parallelization of metaheuristics for phylogenetic inference. Due to the complexity of the problem, the development of parallel algorithms to conduct phylogenetic analyses represents a key challenge to address time-consuming inferences on real biological data sets. In order to provide some guidelines on how the integration of high perfor-

mance computing techniques into our metaheuristic design can be performed, we have used MO-FA and NSGA-II to identify the most computationally demanding operations in their designs. With the information provided by this time profile analysis, two main parallelization approaches have been studied. Firstly, OpenMP-based designs have been proposed to parallelize our metaheuristics on shared-memory multicore multiprocessor systems, studying two different schemes: synchronous generational designs and asynchronous non-generational approaches. In the synchronous generational designs, the main sections that compose each generation of the algorithms are parallelized via OpenMP directives, in such a way that the next generation does not take place until all the tasks in the previous one have been completed. On the other hand, the asynchronous behaviour of honey bees has been used to model an asynchronous nature-inspired approach to parallelize MOABC, in which execution threads are modelled as asynchronous independent entities which perform parallel tasks continuously, dismissing the concept of generation in the metaheuristic.

The second parallel trend studied in this chapter consists of the application of cluster computing techniques to define mixed mode implementations for MO-FA, NSGA-II, and IBEA. Following master-worker guidelines, MPI+OpenMP approaches have been proposed where the master process carries out algorithmic-specific operations, while worker processes carry out the most time-demanding operations identified in the profile analyses. In this way, a two-level parallel design has been described. At the inter-node level, MPI processes are defined with the aim of distributing computations via message-passing techniques. At the intra-node level, OpenMP directives are used to process the assigned tasks by exploiting multithreading capabilities. The performance achieved by this kind of hybrid schemes will depend on three main factors: the parallel efficiency achieved at worker tasks processing, the effect of communication/synchronization patterns as potential overhead sources, and other processing times due to serial components and master tasks.

Chapter 8

Performance Assessment of Parallel Metaheuristics

This chapter details the experimental performance assessment of parallel bioinspired metaheuristics for phylogenetic reconstruction. For this purpose, different parallel metrics are applied to discuss how the studied metaheuristics benefit from the parallel capabilities offered by modern hardware platforms. First of all, a comparative study among synchronous generational designs for the six metaheuristics is conducted, proceeding afterwards with the analysis of asynchronous designs to determine what kind of parallelization schemes leads to the best exploitation of shared-memory multicore systems. Secondly, we address the assessment of mixed mode implementations for multicore clusters and constellation systems, evaluating the effects of the different performance factors which govern the behaviour of hybrid systems. In addition, comparisons with parallel implementations of well-known phylogenetic tools are introduced to verify the quality of our parallel results.

8.1 Initial Considerations

This section describes the different experimental scenarios used to assess the studied parallel bioinspired algorithms. In order to examine the quality of the observed parallel results, two main performance metrics have been considered: speedup and efficiency. The idea is to study the evolution of speedups (with regard to the serial execution time of each metaheuristic) on increasing system sizes, analyzing different problem sizes given by the complexity of the input alignment. In this way, the parallel scalability of the proposals can be evaluated, calculating efficiencies to understand how our parallel designs are able to take advantage of the underlying hardware resources to reduce execution time when analyzing real biological alignments.

In accordance with the different parallel technologies studied in this Thesis, two main evaluation scenarios are considered. The first one deals with the assessment of shared-memory parallelization schemes. In a first step, we evaluate the parallel performance achieved by the synchronous generational designs of our six metaheuristics, in order to analyze which metaheuristic designs achieve a

more efficient exploitation of multicore multiprocessors with OpenMP. The multicore platform used in this first study includes two AMD Opteron Magny-Cours 6174 processors (12 cores per processor) and 32GB RAM (full hardware descriptions and details on other experimental conditions can be found in Chapter 4 - Section 4.4). For experimentation purposes, we have used four benchmark data sets: *rbcL_55*, *mtDNA_186*, *RDPII_218*, and *ZILLA_500*. The configuration of input parameters for the six metaheuristics involves the optimal values identified in our parametric studies (Table 6.3 in Chapter 6), assigning as population size the nearest value to the optimal one which is divisible by the number of cores in the system. Table 8.1 shows the median serial times from 11 independent runs (in seconds) which are used as reference to compute the considered parallel metrics.

An additional case study on shared-memory approaches involves the comparison of synchronous generational schemes and asynchronous non-generational approaches by using MOABC. In this case, we are interested in analyzing which approach leads to the best exploitation of multicore resources, while also checking if the use of different parallelization schemes has an impact on the search capabilities of the metaheuristic. The hardware configuration used in this study comprises four AMD Opteron Magny-Cours 6174 processors (a total of 48 cores) with 64GB RAM. To provide a more detailed discussion on these two metaheuristic parallelization trends, the experimentation has been extended by considering the six data sets used in the assessment of multiobjective and biological results: *rbcL_55*, *mtDNA_186*, *HIV1_192*, *RDPII_218*, *S1482_346*, and *ZILLA_500*. Table 8.2 gives account of the median serial times per dataset considered in this analysis.

Our second experimental scenario addresses the evaluation of mixed mode approaches based on MPI+OpenMP for multicore clusters. For this purpose, we conduct a comparison between an evolutionary design with static workload (NSGA-II) and a swarm-based design with dynamic workload (MO-FA). The idea is to study the relevance of the parallel results achieved by these two approaches, discussing the influence of different performance factors (overhead sources, worker efficiency, master times, etc.) on the observed execution times. These experiments were performed on a cluster configuration composed of six computing nodes, each one containing two quad-core Intel Xeon CPU E5410 processors with 8GB RAM (a total of 48 processing cores). In order to make more noticeable the differences related to the algorithmic features of MO-FA and NSGA-II, this study considers a number of generations = 100 as stop criterion, as well as a common population size = 100 for both algorithms. In this scenario, the computation of parallel metrics for our six nucleotide data sets is performed with regard to the median serial times shown in Table 8.3.

An extension of this mixed mode experimental analysis examines the results obtained by the hybrid version of IBEA over a constellation-like system composed of two interconnected nodes, each one with two AMD Opteron Magny-Cours 6174 processors (48 cores in the complete system) and 32GB RAM. The main goal is to decide which configuration of MPI processes and OpenMP threads leads to an optimal exploitation of this architecture, while also checking the performance of a hybrid approach to parallelize an indicator-based algorithm. These experiments considered our six usual data sets, configuring IBEA with the same input parameters and stop criterion as in the evaluation of shared-memory approaches. Table 8.4 provides IBEA median serial times on this system.

Table 8.1 Serial times in seconds for the study of synchronous approaches on shared-memory systems

Dataset	MOABC	MO-FA	NSGA-II	SPEA2	IMOBA	IBEA
rbcL_55	5610.643	5486.042	5294.762	5355.617	5580.623	5372.389
mtDNA_186	45379.134	47700.121	47135.498	47313.750	46303.295	48083.640
RDPII_218	48218.176	54507.410	50452.111	51045.568	49661.391	51334.486
ZILLA_500	65459.835	77365.742	69702.120	70377.894	68966.945	71307.039

Table 8.2 Serial times in seconds for the comparison of synchronous and asynchronous approaches on shared-memory systems

Method	*rbcL_55*	*mtDNA_186*	*HIV1_192*	*RDPII_218*	*S1482_346*	*ZILLA_500*
MOABC	5610.643	45379.134	14959.778	48218.176	35155.833	65459.835

Table 8.3 Serial times in seconds for the comparison of mixed mode designs on multicore clusters

Method	*rbcL_55*	*mtDNA_186*	*HIV1_192*	*RDPII_218*	*S1482_346*	*ZILLA_500*
MO-FA	5024.666	35402.976	8782.048	25913.257	30299.941	62833.749
NSGA-II	5626.892	42011.738	16236.087	46230.947	37497.891	75826.971

Table 8.4 Serial times in seconds for the evaluation of mixed mode designs on constellation systems

Method	*rbcL_55*	*mtDNA_186*	*HIV1_192*	*RDPII_218*	*S1482_346*	*ZILLA_500*
IBEA	5372.389	48083.640	15494.006	51334.486	36667.600	71307.039

Table 8.5 Serial times in seconds for the parallel biological methods

AMD Opteron 6174-based architecture						
Method	*rbcL_55*	*mtDNA_186*	*HIV1_192*	*RDPII_218*	*S1482_346*	*ZILLA_500*
RAxML	5645.707	45361.424	14561.832	49472.281	35007.067	65190.989
IQPNNI	5534.594	44879.664	15844.212	48763.139	34986.427	68575.510
MrBayes	5505.500	47160.020	15082.840	48324.990	34854.760	65729.110
Intel Xeon E5410-based architecture						
Method	*rbcL_55*	*mtDNA_186*	*HIV1_192*	*RDPII_218*	*S1482_346*	*ZILLA_500*
RAxML	5643.685	42201.759	16847.988	47072.763	36466.515	75460.527
IQPNNI	5455.805	42266.978	15971.191	47182.938	36170.262	74756.936
MrBayes	5579.790	42712.910	16711.579	45477.494	38193.503	76842.730

All the studied hardware configurations run under a Scientific Linux OS with GCC compiler and MPICH as MPI implementation, enabling the -O3 optimization flag for software compilation purposes. The stochastic nature of bioinspired algorithms and the variability in execution times introduced by well-known hardware/software factors (such as operating system process scheduling and other background tasks) must be taken into account to provide statistical robustness to the comparisons. To achieve this, each experiment involves 31 independent runs per system/problem size. The evaluation of speedup samples also considers the application of our statistical methodology (Chapter 4 - Section 4.5) to check for statistically significant differences in parallel results.

Finally, the relevance of our parallel results will be examined by conducting comparisons with state-of-the-art parallel tools for phylogenetics: RAxML [171], IQPNNI [120], and MrBayes [148]. RAxML includes parallel releases based on POSIX threads and hybrid MPI+POSIX to conduct maximum likelihood-based phylogenetic analyses. The main strategy implemented in this software follows a shared-memory parallelization of likelihood computations via POSIX threads, using MPI in the hybrid version to distribute multiple phylogenetic searches among MPI processes. On the other hand, IQPNNI counts with pure OpenMP and hybrid MPI+OpenMP versions based on master-worker models to parallelize topological searches and evaluations. Finally, MrBayes provides pure OpenMP and hybrid MPI+OpenMP versions to deal with computationally-demanding operations in its Monte Carlo design. To make fair comparisons, the input parameters for each method were configured in accordance with the serial execution times reported by our metaheuristics. Table 8.5 details the serial times reported by these parallel tools on the two processor architectures considered in this study. These times will be used to assess the parallel scalability shown by each biological tool, so that our comparisons will be conducted by using the observed speedups as reference. In addition, comparisons of speedups with the MPI and MPI+OpenMP versions of the multiobjective tool PhyloMOEA will also be introduced, considering the results up to 16 cores reported in [25].

8.2 Evaluation of Shared-Memory Approaches

This section introduces the experimental assessment of shared-memory approaches to parallelize our metaheuristics. We focus firstly on conducting comparisons among the studied algorithms under synchronous parallelization schemes, undertaking afterwards the evaluation of asynchronous designs.

8.2.1 Synchronous Metaheuristic Designs

With the aim of analyzing the parallel performance achieved by our OpenMP-based synchronous designs, we have performed experiments considering increasing system sizes involving 4, 8, 16, and 24 execution cores. The parallel times reported by each metaheuristic were examined by computing speedups and efficiencies over the serial times described in Table 8.1. Table 8.6 shows the median speedup values (SU) and efficiencies (Eff in %) achieved by MOABC (columns 2-3), MO-FA (4-5), NSGA-II (6-7), SPEA2 (8-9), IMOBA (10-11), and IBEA (12-13) on the four real benchmark data sets considered in our experiments. Statistical pairwise comparisons of parallel performance are presented in Table 8.7, where we show the results of applying our statistical methodology over the speedup samples obtained when using the entire hardware platform (24 cores). In addition, Figures 8.1 and 8.2 provide graphical representations of parallel scalability for each design, showing the evolution of speedups with regard to the theoretical linear speedup.

Attending to these results, three main groups of algorithmic designs can be distinguished in terms of parallel performance. The first group contains MO-FA as the leading algorithm, reporting the best speedups and efficiencies in all the system and problem sizes under study. The OpenMP version of

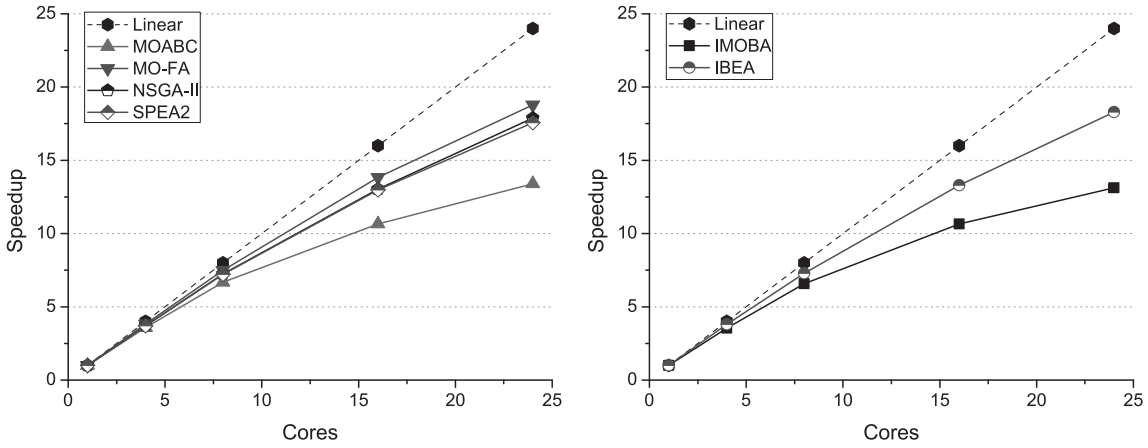
Table 8.6 Synchronous generational designs - speedups and efficiencies

rbcL_55												
	MOABC		MO-FA		NSGA-II		SPEA2		IMOBA		IBEA	
Cores	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)
4	3.510	87.738	**3.690**	**92.248**	3.640	91.012	3.636	90.903	3.432	85.802	3.662	91.557
8	6.629	82.858	**7.206**	**90.079**	6.831	85.393	6.873	85.912	6.465	80.813	6.950	86.869
16	10.829	67.684	**13.217**	**82.607**	12.279	76.745	12.157	75.978	10.856	67.853	12.321	77.005
24	14.342	59.758	**17.574**	**73.226**	16.772	69.884	15.992	66.633	13.773	57.385	16.834	70.142
mtDNA_186												
	MOABC		MO-FA		NSGA-II		SPEA2		IMOBA		IBEA	
Cores	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)
4	3.602	90.061	**3.867**	**96.668**	3.662	91.542	3.618	90.454	3.585	89.626	3.833	95.834
8	6.686	83.572	**7.453**	**93.161**	7.207	90.095	7.170	89.621	6.684	83.552	7.210	90.122
16	10.587	66.170	**13.402**	**83.763**	12.405	77.533	12.346	77.164	10.910	68.189	12.897	80.604
24	12.944	53.935	**17.880**	**74.500**	17.084	71.184	16.542	68.924	13.207	55.029	17.562	73.173
RDPII_218												
	MOABC		MO-FA		NSGA-II		SPEA2		IMOBA		IBEA	
Cores	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)
4	3.626	90.652	**3.892**	**97.320**	3.695	92.376	3.651	91.284	3.606	90.157	3.858	96.439
8	6.949	86.869	**7.509**	**93.866**	7.277	90.960	7.153	89.418	6.753	84.408	7.297	91.207
16	10.827	67.671	**13.576**	**84.849**	13.177	82.355	13.093	81.832	10.498	65.615	13.369	83.558
24	13.154	54.809	**18.062**	**75.260**	17.425	72.604	17.490	72.875	12.772	53.218	18.007	75.031
ZILLA_500												
	MOABC		MO-FA		NSGA-II		SPEA2		IMOBA		IBEA	
Cores	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)
4	3.623	90.572	**3.893**	**97.324**	3.857	96.443	3.849	96.231	3.517	87.916	3.867	96.675
8	6.418	80.221	**7.764**	**97.050**	7.650	95.622	7.642	95.526	6.405	80.060	7.684	96.054
16	10.373	64.829	**15.113**	**94.455**	14.260	89.127	14.289	89.306	10.350	64.686	14.572	91.075
24	13.147	54.777	**21.589**	**89.952**	20.254	84.391	20.222	84.257	12.714	52.977	20.726	86.358

Table 8.7 Statistical testing of speedup samples (✓=significant differences, ×=non-significant)

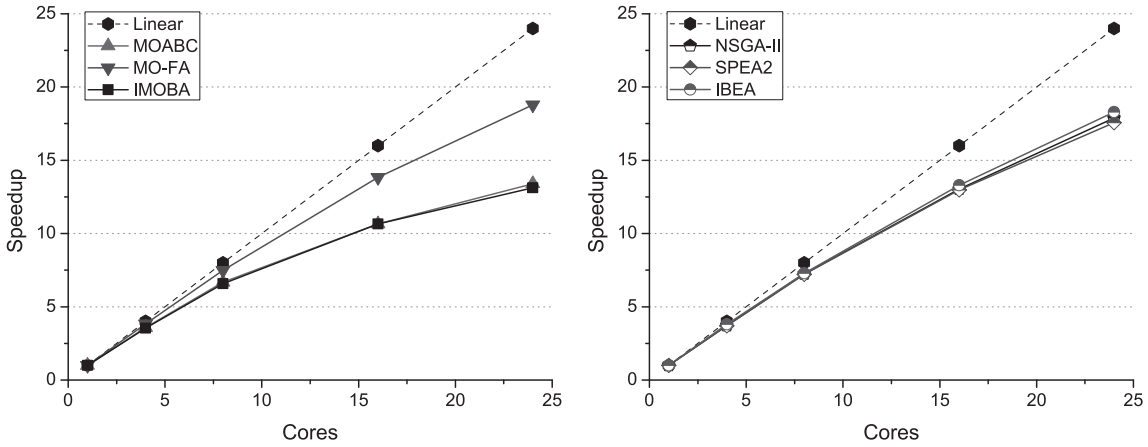
	rbcL_55						*mtDNA_186*					
Algorithm	MOABC	MO-FA	NSGA-II	SPEA2	IMOBA	IBEA	MOABC	MO-FA	NSGA-II	SPEA2	IMOBA	IBEA
MOABC		✓	✓	✓	✓	✓		✓	✓	✓	×	✓
MO-FA			✓	✓	✓	✓			✓	✓	✓	✓
NSGA-II				✓	✓	×				✓	✓	✓
SPEA2					✓	✓					✓	✓
IMOBA						✓						✓
IBEA												
	RDPII_218						*ZILLA_500*					
Algorithm	MOABC	MO-FA	NSGA-II	SPEA2	IMOBA	IBEA	MOABC	MO-FA	NSGA-II	SPEA2	IMOBA	IBEA
MOABC		✓	✓	✓	×	✓		✓	✓	✓	✓	✓
MO-FA			✓	✓	✓	×			✓	✓	✓	✓
NSGA-II				×	✓	✓				×	✓	✓
SPEA2					✓	✓					✓	✓
IMOBA						✓						✓
IBEA												

MO-FA gives rise to speedups in the range 3.69 - 3.89 (for 4 cores), 7.21 - 7.76 (8 cores), 13.22 - 15.11 (16 cores), and 17.57 - 21.59 (24 cores). The second group comprises the reference MOEAs (IBEA, NSGA-II, and SPEA2), which also obtain relevant parallel results in all the considered data sets (only outperformed by MO-FA). In this class, IBEA rises as the best algorithm (the second best one in overall), whose improvement becomes more noticeable when considering configurations involving 24 cores (leading to efficiencies in the range 70.14% - 86.36%). The third group includes IMOBA and MOABC, which show the lowest speedups and efficiencies in all the examined cases.



(a) Dominance-based approaches (b) Indicator-based approaches

Fig. 8.1 Mean speedups comparisons attending to the fitness assignment procedures



(a) Swarm intelligence approaches (b) Evolutionary approaches

Fig. 8.2 Mean speedups comparisons attending to the features of the algorithmic design

Therefore, the evaluation of parallel results points out MO-FA as the synchronous algorithm which attains the most satisfying exploitation of shared-memory resources to reduce execution time. This statement is supported by the statistical testing of speedups in Table 8.7, which gives account of the statistical significant improvement achieved by MO-FA over the remaining methods in almost all the cases (only non-significant differences are observed with regard to IBEA on *RDPII_218*). These results suggest the relevance of the proposed parallel design and the effectiveness of the load balance mechanisms and structures introduced to support the firefly movement step.

Regarding the reference evolutionary designs, it can be observed how IBEA, NSGA-II, and SPEA2 achieve similar parallel scalability, due to the fact that they share a common main parallelizable fraction of code (given by the offspring generation loop). The differences observed in these algorithms are motivated by their different serial time percentages (due to fitness assignments, environmental selections, and other operations). For example, the time percentage spent on serial

computations by SPEA2 is, in average, 2.5 times higher than the percentage spent by NSGA-II. On the other hand, the more time-demanding computation of indicator-based fitness values in IBEA is able to benefit from multicore capabilities, as indicator calculations can be performed in parallel. As a result, IBEA is able to achieve a statistically significant improvement over NSGA-II and SPEA2 on *mtDNA_186*, *RDPII_218*, and *ZILLA_500* when using the 24 processing cores available in the shared-memory configuration under study.

A remarkable feature shown by the best four parallel metaheuristics is given by the speedup improvement observed over growing problem sizes (in terms of number of species in the input alignment). Focusing on MO-FA, the speedups achieved by this algorithm rise from 17.574 (for the smallest dataset *rbcL_55*) to 17.880 / 18.062 (for the medium-sized data sets *mtDNA_186* and *RDPII_218*) and a significant 21.589 (for the dataset with the most species *ZILLA_500*). These results can be explained in the following terms. By considering growing numbers of sequences in the input dataset, the generation and evaluation of new candidate solutions will involve more computations over growing floating-point distance matrices and tree topologies. As these operations take place inside parallel regions defined by *#pragma omp for* directives, an increase in the parallelizable fraction of the application is attained, leading to better parallel results.

On the other hand, the experimentation on IMOBA and MOABC gives account of the weaker scalability obtained by their synchronous implementations in comparison to the remaining methods. Both metaheuristics show some characteristics in their algorithmic designs which make difficult the attainment of a full exploitation of shared-memory multicore systems by using a synchronous parallelization approach. For example, the scout bee step in MOABC involves a variable amount of scout searches (as many as exhausted solutions have been detected in the current generation). If this number of scout searches is not high enough to fit the number of execution threads, the computation of the scout step can lead to wasted multithreading resources (idle threads which must wait for the other ones until scout tasks have been completed according to the synchronous principle). As for IMOBA, the bat search step is governed by an if-else condition which determines if exploitations or explorations must be conducted. In spite of using dynamic scheduling policies, the different computational complexities shown by these two procedures still lead to noticeable load imbalance issues (and, therefore, thread waiting times), resulting into more modest parallel results. In fact, our statistical tests confirm the similar behaviour of MOABC and IMOBA, which show non-significant differences in speedups for *mtDNA_186* and *RDPII_218*. These issues motivate the research on alternative parallelization schemes for this kind of algorithms. Using MOABC as case study, we address next the comparison of this synchronous design with our asynchronous nature-inspired proposal.

8.2.2 Comparative Assessment of Synchronous and Asynchronous Designs

In order to evaluate the proposed synchronous (SG-MOABC) and asynchronous (AN-MOABC) parallel implementations of MOABC, we have examined the parallel results obtained in executions involving 8, 16, 24, 32, and 48 processing cores. Considering the serial times in Table 8.2, Table 8.8

Table 8.8 Comparisons of parallel performance between SG-MOABC and AN-MOABC

	rbcL_55				*mtDNA_186*				*HIV1_192*			
	SG-MOABC		AN-MOABC		SG-MOABC		AN-MOABC		SG-MOABC		AN-MOABC	
Cores	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)
8	6.629	82.858	**7.235**	**90.432**	6.686	83.572	**7.243**	**90.537**	6.461	80.768	**7.221**	**90.265**
16	10.829	67.684	**14.464**	**90.398**	10.587	66.170	**14.455**	**90.341**	10.347	64.671	**14.420**	**90.125**
24	14.342	59.758	**21.639**	**90.161**	12.944	53.935	**21.636**	**90.151**	13.552	56.465	**21.626**	**90.107**
32	14.950	46.720	**28.823**	**90.072**	15.290	47.781	**28.835**	**90.109**	14.582	45.568	**28.801**	**90.002**
48	18.222	37.963	**40.694**	**84.779**	19.383	40.381	**40.850**	**85.104**	19.432	40.483	**42.516**	**88.575**
	RDPII_218				*S1482_346*				*ZILLA_500*			
	SG-MOABC		AN-MOABC		SG-MOABC		AN-MOABC		SG-MOABC		AN-MOABC	
Cores	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)
8	6.949	86.869	**7.260**	**90.745**	6.861	85.763	**7.509**	**93.859**	6.418	80.221	**7.532**	**94.156**
16	10.827	67.671	**14.501**	**90.632**	11.289	70.557	**14.982**	**93.639**	10.373	64.829	**15.052**	**94.072**
24	13.154	54.809	**21.627**	**90.111**	14.520	60.498	**22.438**	**93.494**	13.147	54.777	**22.568**	**94.034**
32	14.347	44.835	**28.812**	**90.037**	15.290	47.783	**29.670**	**92.719**	14.048	43.899	**30.060**	**93.937**
48	16.936	35.283	**42.864**	**89.301**	20.575	42.864	**43.235**	**90.074**	18.363	38.256	**44.761**	**93.252**

details the comparison of median speedups and efficiencies between SG-MOABC (columns 2-3, 6-7, and 10-11) and AN-MOABC (columns 4-5, 8-9, and 12-13) over the six data sets under study.

Table 8.8 denotes that AN-MOABC achieves a significant improvement over SG-MOABC for all the considered problem and system sizes. In fact, the results of applying our statistical methodology over the speedup samples achieved by the two parallel designs pointed out that AN-MOABC leads to a statistically significant improvement over SG-MOABC in all the studied configurations. The obtained efficiency values suggest that the asynchronous model attains an almost optimal use of hardware resources. On the other hand, the decreasing efficiencies observed for the synchronous design suggest a worsening in the capability of the algorithm to exploit the shared-memory environment. When using all the 48 cores in the architecture, the asynchronous non-generational model is able to give rise to efficiencies from 84.779% to 93.252%, in comparison with the 35.283% - 42.864 % efficiency range obtained by the synchronous generational model. In conclusion, the asynchronous parallelization scheme in AN-MOABC shows very good scalability (as opposed to SG-MOABC).

In order to discuss these results, we must bear in mind the algorithmic features of each parallel scheme. In SG-MOABC, the synchronous model motivates that all the parallel tasks to be performed in each section must finish before processing the tasks from the next section. Furthermore, waiting times are also introduced when sequential operations enclosed by *#pragma omp single* directives must be performed. Hence, the SG-MOABC parallel time can be formulated in the following way:

$$T_{total}(SG-MOABC) = T_{employed\_step} + T_{onlooker\_step} + T_{scout\_step} + T_{seq\_op}, \tag{8.1}$$

where $T_{employed\_step}$ refers to the accumulated parallel times spent on performing the employed bee step at each generation, $T_{onlooker\_step}$ the accumulated times for the onlooker bee step, T_{scout_step} the times corresponding to scout bees searches, and T_{seq_op} the times spent on sequential operations.

On the other hand, AN-MOABC is based on the assumption that worker threads must perform their tasks without waiting for the remaining workers to complete their workload. Once they have

Table 8.9 Analysis of parallel times (in seconds) for SG-MOABC

	rbcL_55					*mtDNA_186*				
Cores	Temployed step	Tonlooker step	Tscout step	Tseq op	(Eq. 8.1) Ttotal	Temployed step	Tonlooker step	Tscout step	Tseq op	(Eq. 8.1) Ttotal
8	362.54	360.48	122.75	0.65	**846.42**	2963.80	2946.06	875.53	2.06	**6787.45**
16	201.10	200.41	115.91	0.68	**518.10**	1710.10	1699.27	875.91	2.10	**4287.38**
24	137.82	137.66	115.04	0.69	**391.20**	1372.66	1371.38	759.60	2.05	**3505.69**
32	131.59	127.38	115.70	0.61	**375.28**	1049.10	1045.51	871.28	2.03	**2967.94**
48	95.73	94.76	116.78	0.66	**307.92**	722.55	719.93	896.67	2.04	**2341.19**
	HIV1_192					*RDPII_218*				
8	993.06	968.72	351.37	2.10	**2315.25**	2728.85	2697.70	1509.33	2.50	**6938.38**
16	535.90	524.05	383.63	2.17	**1445.75**	1504.59	1497.01	1449.23	2.55	**4453.38**
24	379.34	370.64	351.78	2.15	**1103.90**	1157.08	1114.76	1391.22	2.57	**3665.63**
32	349.61	341.29	332.89	2.12	**1025.91**	991.90	967.53	1391.66	2.59	**3353.69**
48	216.31	211.16	340.22	2.16	**769.85**	794.34	724.63	1325.58	2.56	**2847.12**
	S1482_346					*ZILLA_500*				
8	2191.08	2151.07	777.19	4.65	**5124.00**	4006.28	3969.28	2215.68	8.73	**10199.98**
16	1251.60	1234.64	623.27	4.63	**3114.13**	2144.41	2129.08	2028.56	8.75	**6310.80**
24	831.88	818.42	766.20	4.62	**2421.12**	1487.06	1473.95	2009.49	8.76	**4979.26**
32	772.83	760.96	760.82	4.60	**2299.12**	1376.74	1368.48	1906.05	8.57	**4659.84**
48	515.35	486.57	702.13	4.61	**1708.67**	841.22	826.09	1888.90	8.54	**3564.75**

finished, new tasks are processed immediately after introducing the results of the previous search into the queues. These operations are repeated until the termination signal from the master is received. Therefore, the total time spent by AN-MOABC on completing a run can be expressed as:

$$T_{total}(AN-MOABC) = \max(T_{employed\_worker}, T_{onlooker\_worker}, T_{master}), \tag{8.2}$$

where $T_{employed\_worker}$ represents the time spent by the slowest employed worker (which includes the times needed to compute employed bee tasks and scout bee tasks), $T_{onlooker\_worker}$ refers to the time spent by the slowest onlooker worker (including the times required by onlooker bee tasks and scout bee tasks), and T_{master} corresponds to the master thread times (comprising swarm update times, solution acceptance times, and those times where the master waits for new solutions from the workers).

Using these equations, Tables 8.9 and 8.10 introduce the analysis of parallel times for SG-MOABC and AN-MOABC. These times were taken from the executions which achieved the median speedups from Table 8.8. For each dataset and system configuration, Table 8.9 provides the values in seconds for $T_{employed\_step}$ (columns 2 and 7), $T_{onlooker\_step}$ (columns 3 and 8), T_{scout_step} (columns 4 and 9), and T_{seq_op} (columns 5 and 10). The result of applying Equation 8.1 over these times is given by columns 6 and 11. Regarding Table 8.10, we can find the times for the slowest employed worker (columns 2-4) and onlooker worker (columns 5-7), as well as master times (column 8). In this table, T_{total} (column 9) defines the value resulting of applying Equation 8.2 over these previous times.

By analyzing Table 8.9, two main factors which explain the decreased performance obtained by the synchronous implementation in SG-MOABC can be identified. First of all, in spite of using a dynamic scheduling mechanism in the worksharing directives, the synchronous parallelization of the employed and onlooker loops is affected by those system configurations in which a proper

Table 8.10 Analysis of parallel times (in seconds) for AN-MOABC

	Employed Worker Times			*Onlooker Worker Times*			*Master*	
Cores	Temployed tasks	Tscout tasks	Temployed worker	Tonlooker tasks	Tscout tasks	Tonlooker worker	*Times* Tmaster	(Eq. 8.2) Ttotal
				rbcL_55				
8	759.41	16.06	775.47	764.37	9.70	774.07	773.96	**775.47**
16	380.91	7.00	387.91	384.96	2.95	387.91	387.22	**387.91**
24	252.22	6.42	258.64	258.79	0.49	259.27	258.51	**259.27**
32	188.23	6.43	194.65	191.92	1.14	193.06	192.32	**194.65**
48	133.62	4.25	137.87	137.25	0.50	137.74	137.13	**137.87**
				mtDNA_186				
8	6030.80	234.45	6265.24	6211.49	45.14	6256.63	6253.65	**6265.24**
16	3057.84	81.57	3139.41	3101.48	36.57	3138.06	3131.88	**3139.41**
24	2071.57	25.78	2097.36	2089.36	7.53	2096.89	2092.51	**2097.36**
32	1547.16	26.60	1573.75	1557.62	7.29	1564.92	1561.30	**1573.75**
48	1087.76	23.11	1110.87	1096.88	8.30	1105.18	1099.15	**1110.87**
				HIV1_192				
8	1966.01	105.53	2071.54	2040.87	24.14	2065.01	2063.55	**2071.54**
16	999.58	37.83	1037.41	1025.30	10.58	1035.88	1035.00	**1037.41**
24	672.84	18.88	691.72	677.25	13.86	691.11	687.11	**691.72**
32	510.05	9.36	519.41	512.82	5.61	518.43	514.73	**519.41**
48	329.71	22.15	351.86	348.15	1.88	350.03	348.00	**351.86**
				RDPII_218				
8	6228.89	413.13	6642.01	6350.93	270.74	6621.67	6615.97	**6642.01**
16	3115.04	210.32	3325.36	3244.95	75.68	3320.63	3314.61	**3325.36**
24	2083.26	146.17	2229.43	2222.56	7.00	2229.55	2224.06	**2229.55**
32	1649.09	24.46	1673.55	1665.22	7.14	1672.36	1667.32	**1673.55**
48	1086.51	38.37	1124.88	1081.94	6.81	1088.75	1081.39	**1124.88**
				S1482_346				
8	4577.83	103.72	4681.54	4625.09	56.92	4682.01	4677.55	**4682.01**
16	2284.30	62.18	2346.49	2319.34	24.58	2343.92	2341.87	**2346.49**
24	1555.41	11.36	1566.76	1558.81	7.75	1566.55	1562.38	**1566.76**
32	1170.65	14.22	1184.87	1181.16	3.20	1184.36	1180.67	**1184.87**
48	798.77	13.93	812.69	809.52	3.61	813.13	808.36	**813.13**
				ZILLA_500				
8	8236.26	454.09	8690.35	8453.09	236.61	8689.70	8685.21	**8690.35**
16	4219.59	129.16	4348.75	4258.07	88.60	4346.67	4342.58	**4348.75**
24	2663.91	236.63	2900.54	2824.49	68.66	2893.15	2885.49	**2900.54**
32	2088.41	89.22	2177.63	2087.46	46.97	2134.42	2125.06	**2177.63**
48	1234.88	227.54	1462.42	1398.73	47.56	1446.29	1438.07	**1462.42**

Table 8.11 Multiobjective quality comparison between AN-MOABC and SG-MOABC

Hypervolume							
Dataset	AN-MOABC	SG-MOABC	Sign?	Dataset	AN-MOABC	SG-MOABC	Sign?
rbcL_55	71.73±0.02	71.74±0.02	×	*RDPII_218*	74.64±0.05	74.65±0.04	×
mtDNA_186	70.02±0.01	70.02±0.02	×	*S1482_346*	80.80±0.15	80.79±0.17	×
HIV1_192	88.89±0.27	88.91±0.33	×	*ZILLA_500*	73.02±0.01	73.02±0.01	×

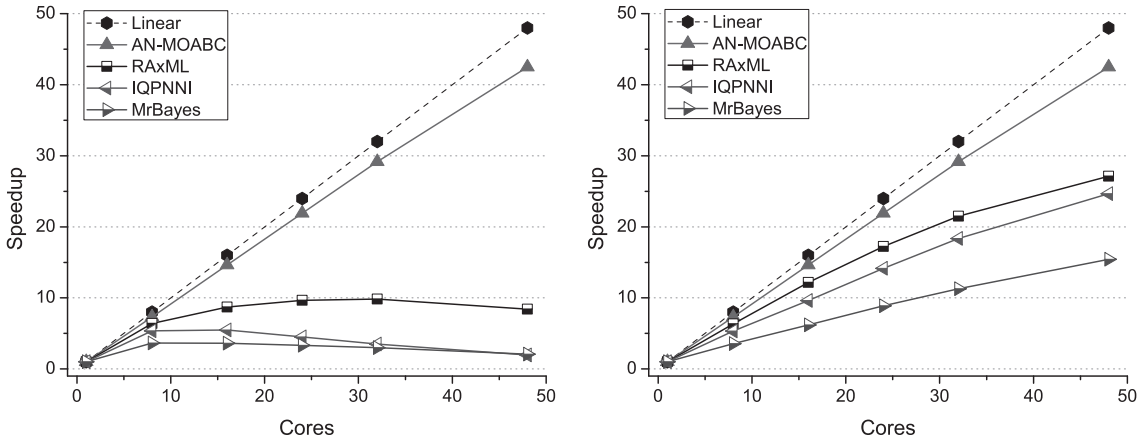
distribution of iterations per thread cannot be applied. When the *popSize/2* bees cannot be equally distributed among *num_threads* threads, additional iterations of the employed/onlooker loop are re-

quired to deal with the remaining *(popSize/2)%num_threads* bees. When computing these additional tasks, the remaining threads keep waiting inside the implicit barrier located at the end of *#pragma omp for* directives, giving as a result idle multithreading resources. In Table 8.9, this issue can be observed when comparing the times reported by system sizes involving 24 and 32 cores.

The second performance issue shown by the synchronous generational design is given by the scout bee step, as suggested previously in Section 8.2.1. When examining the columns 4 and 9 in Table 8.9, we cannot observe a significant improvement in scout times when using an increasing number of cores. In order to explain this issue, we must consider that exhausted solutions in the swarm are replaced by scout solutions as soon as they are detected, leading to a variable number of iterations in the scout loop which may not fit the number of execution threads. Hence, all those threads that do not contribute to the computation of scout searches remain idle. The worst case scenario is given by those situations where only a single solution has reached the limit control parameter. For configurations of 48 cores, the synchronous generational design assigns this scout bee search to one execution thread, while the remaining 47 threads must wait for the termination of this task. Therefore, the scout bee step represents a meaningful bottleneck in SG-MOABC, as all the threads must wait for the termination of scout searches before carrying out new employed/onlooker tasks.

On the other hand, Table 8.10 suggests that the asynchronous proposal in AN-MOABC is not affected by such performance issues, leading to an effective reduction in T_{total} as we increase the number of cores. The main reason which explains this improvement is given by the fact that worker threads are able to carry out parallel tasks continuously, without requiring any waiting times besides the ones needed to ensure data integrity with the master thread. Therefore, the asynchronous nature of AN-MOABC solves the first issue shown by SG-MOABC, as new worker tasks can be computed without waiting for the remaining workers. Regarding scout searches, these tasks do not represent either a performance pitfall. When a scout search is required, the remaining worker threads can compute parallel tasks while the scout task is being processed by the thread which found the exhausted solution. It should be noted that the variability shown in scout tasks times is due to the stochastic nature of MOABC, as different executions may require different numbers of scout searches.

These results show the benefits of applying asynchronous non-generational schemes to parallelize metaheuristics. Such novel approaches are able to deal with the main performance issues observed in traditional synchronous designs, making an efficient use of resources in a shared-memory environment. By exploiting such architectures with AN-MOABC, the execution times required to conduct a multiobjective analysis on a complex dataset like *ZILLA_500* can be reduced from 18.18 hours to 24.27 minutes. As a result, AN-MOABC becomes the leading parallel approach, outperforming not only SG-MOABC, but also the remaining five parallel metaheuristics in terms of parallel scalability. Moreover, Table 8.11 supports the idea that our asynchronous parallel design is also able to keep the search capabilities of the original MOABC generational design, according to the non-significant differences in hypervolume scores obtained with regard to SG-MOABC (columns *Sign?*). Therefore, the asynchronous version of MOABC leads to an efficient exploitation of shared-memory multicore systems while preserving the multiobjective quality of the inferred phylogenies.



(a) Multithreaded tools (b) Hybrid tools

Fig. 8.3 Mean speedups comparisons with other parallel tools on shared-memory platforms

8.2.3 Comparisons with Other Parallel Phylogenetic Methods

Now we undertake the assessment of our best parallel approach for shared-memory systems (AN-MOABC) with regard to state-of-the-art parallel methods for phylogenetic inference. For these purpose, we have made comparisons with the POSIX/OpenMP multithreaded versions of RAxML, IQPPNI, and MrBayes, as well as with their hybrid MPI+POSIX/MPI+OpenMP versions. For each parallel trend, Tables 8.12 and 8.13 compare the median speedups and efficiencies achieved by AN-MOABC (columns 2-3), RAxML (columns 4-5), IQPNNI (columns 6-7), and MrBayes (8-9). A graphical representation of the mean speedups observed for each parallel method is given in Figure 8.3. The results of our experiments suggest that AN-MOABC leads to significant parallel scalability on increasing number of cores, improving the speedups and efficiencies reported by the multicore and hybrid parallel versions of the biological methods. Moreover, the statistical testing of speedups gave account of statistically significant differences between AN-MOABC and the biological methods in almost all the considered problem and system sizes (non-significant differences were only found with regard to RAxML on *mtDNA_186* in configurations involving 8 cores).

Table 8.12 verifies the key role that the exploitation of multithreading resources plays in the way the proposals scale on increasing system sizes. Focusing on the multicore versions of RAxML, IQPNNI, and MrBayes, slight improvements in speedup are attained by these methods for configurations involving more than 16 cores. In fact, a worsening is observed for configurations of 24, 32 and 48 cores for IQPNNI and MrBayes and 48 cores for RAxML. The POSIX/OpenMP intra-algorithm parallelization used in these methods mainly focuses on accelerating the computation of the objective function to minimize the times required to assess candidate phylogenies. According to our results, the multithreading efficiency of such approaches can be affected by thread management times and sequential times where the algorithms are inferring the phylogeny to be evaluated. The introduction of MPI to support these multicore schemes helps to improve the scalability observed on increasing number of cores (Table 8.13), although the shared-memory parallelization strategies im-

Table 8.12 Comparisons between AN-MOABC and other parallel methods (multithreaded versions)

	rbcL_55							
	AN-MOABC		RAxML-Multicore		IQPNNI-Multicore		MrBayes-Multicore	
Cores	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)
8	**7.235**	**90.432**	6.258	78.225	5.380	67.250	3.385	42.313
16	**14.464**	**90.398**	8.326	52.038	5.138	32.113	2.633	16.456
24	**21.639**	**90.161**	8.775	36.563	4.017	16.738	2.394	9.975
32	**28.823**	**90.072**	8.585	26.828	2.903	9.072	2.078	6.494
48	**40.694**	**84.779**	7.125	14.844	1.537	3.202	1.408	2.933
	mtDNA_186							
8	**7.243**	**90.537**	7.238	90.475	5.030	62.875	5.971	74.638
16	**14.455**	**90.341**	10.389	64.931	5.560	34.750	7.695	48.094
24	**21.636**	**90.151**	12.887	53.696	4.691	19.546	7.692	32.050
32	**28.835**	**90.109**	12.870	40.219	3.733	11.666	7.208	22.525
48	**40.850**	**85.104**	11.236	23.408	2.340	4.875	5.144	10.717
	HIV1_192							
8	**7.221**	**90.265**	6.044	75.550	4.990	62.375	2.440	30.500
16	**14.420**	**90.125**	8.239	51.494	4.507	28.169	2.297	14.356
24	**21.626**	**90.107**	8.174	34.058	3.482	14.508	1.923	8.013
32	**28.801**	**90.003**	8.088	25.275	2.469	7.716	1.625	5.078
48	**42.516**	**88.575**	6.482	13.504	1.263	2.631	1.087	2.265
	RDPII_218							
8	**7.260**	**90.745**	6.538	81.725	6.890	86.125	4.662	58.275
16	**14.501**	**90.632**	9.311	58.194	9.033	56.456	4.135	25.844
24	**21.627**	**90.111**	11.345	47.271	8.338	34.742	4.019	16.746
32	**28.812**	**90.037**	12.538	39.181	6.991	21.847	3.700	11.563
48	**42.864**	**89.301**	12.237	25.494	3.815	7.948	2.645	5.510
	S1482_346							
8	**7.509**	**93.859**	6.284	78.550	4.610	57.625	2.717	33.963
16	**14.982**	**93.639**	8.514	53.213	3.522	22.013	2.428	15.175
24	**22.438**	**93.494**	9.039	37.663	2.606	10.858	1.947	8.113
32	**29.670**	**92.719**	8.675	27.109	1.975	6.172	1.709	5.341
48	**43.235**	**90.074**	6.904	14.383	1.257	2.619	1.112	2.317
	ZILLA_500							
8	**7.532**	**94.156**	5.991	74.888	5.238	65.475	2.683	33.543
16	**15.052**	**94.072**	7.413	46.331	5.142	32.138	2.462	15.388
24	**22.568**	**94.034**	7.721	32.171	3.971	16.546	2.021	8.421
32	**30.060**	**93.937**	8.223	25.697	2.863	8.947	1.708	5.338
48	**44.761**	**93.252**	6.537	13.619	1.545	3.219	1.122	2.338

plemented in these methods still have a noticeable impact in their speedup results. On the other hand, AN-MOABC is able to take advantage of the underlying resources by relying on an asynchronous parallel design which reduces performance problems at the multicore level.

In addition, Table 8.14 introduces a comparison of parallel performance with PhyloMOEA. By examining the parallel results reported by this multiobjective tool [25], we can observe how AN-MOABC obtains significant speedups with regard to the two parallel versions of PhyloMOEA, as a result of the efficient use of shared-memory resources achieved by our asynchronous design. In conclusion, this comparative study with other parallel methods from the literature has shown the

Table 8.13 Comparisons between AN-MOABC and other parallel methods (hybrid versions)

	rbcL_55							
	AN-MOABC		RAxML-Hybrid		IQPNNI-Hybrid		MrBayes-Hybrid	
Cores	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)
8	**7.235**	**90.432**	5.984	74.800	5.376	67.200	3.208	40.100
16	**14.464**	**90.398**	11.485	71.781	8.745	54.656	5.815	36.344
24	**21.639**	**90.161**	17.204	71.683	12.929	53.871	8.168	34.033
32	**28.823**	**90.072**	22.389	69.966	16.917	52.866	10.273	32.103
48	**40.694**	**84.779**	26.643	55.506	23.466	48.888	13.798	28.746
	mtDNA_186							
8	**7.243**	**90.537**	7.219	90.238	5.020	62.750	5.881	73.513
16	**14.455**	**90.341**	14.193	88.706	9.458	59.113	10.442	65.263
24	**21.636**	**90.151**	20.019	83.413	14.025	58.438	15.330	63.875
32	**28.835**	**90.109**	24.904	77.825	18.427	57.584	19.885	62.141
48	**40.850**	**85.104**	30.181	62.877	25.620	53.375	28.444	59.258
	HIV1_192							
8	**7.221**	**90.265**	6.025	75.313	4.983	62.288	2.372	29.650
16	**14.420**	**90.125**	11.776	73.600	9.111	56.944	4.310	26.938
24	**21.626**	**90.107**	15.865	66.104	13.233	55.138	6.232	25.967
32	**28.801**	**90.002**	19.610	61.281	17.420	54.438	8.060	25.188
48	**42.516**	**88.575**	20.941	43.627	22.843	47.590	10.885	22.677
	RDPII_218							
8	**7.260**	**90.745**	6.530	81.625	6.802	85.025	4.635	57.938
16	**14.501**	**90.632**	12.245	76.531	13.310	83.188	7.514	46.963
24	**21.627**	**90.111**	16.786	69.942	19.276	80.317	10.677	44.488
32	**28.812**	**90.037**	21.996	68.738	25.124	78.513	13.192	41.225
48	**42.864**	**89.301**	29.582	61.629	33.259	69.290	17.162	35.754
	S1482_346							
8	**7.509**	**93.859**	6.274	78.425	4.575	57.188	2.618	32.725
16	**14.982**	**93.639**	12.106	75.663	7.729	48.306	4.540	28.375
24	**22.438**	**93.499**	17.490	72.875	11.386	47.442	6.417	26.738
32	**29.670**	**92.719**	20.579	64.309	14.114	44.106	8.073	25.228
48	**43.235**	**90.074**	26.679	55.581	18.884	39.342	11.222	23.379
	ZILLA_500							
8	**7.532**	**94.156**	5.949	74.363	5.201	65.013	2.625	32.813
16	**15.052**	**94.072**	11.249	70.306	9.465	59.156	4.601	28.756
24	**22.568**	**94.034**	16.105	67.104	14.064	58.600	6.434	26.808
32	**30.060**	**93.937**	19.585	61.203	18.233	56.978	8.295	25.922
48	**44.761**	**93.252**	27.008	56.267	23.959	49.915	11.110	23.146

Table 8.14 Comparisons between AN-MOABC and PhyloMOEA (16 cores)

	AN-MOABC		PhyloMOEA MPI		PhyloMOEA Hybrid	
Dataset	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)
rbcL_55	**14.46**	**90.40**	7.30	45.63	8.30	51.88
mtDNA_186	**14.46**	**90.34**	7.40	46.25	8.50	53.13
RDPII_218	**14.50**	**90.63**	9.80	61.25	10.20	63.75
ZILLA_500	**15.05**	**94.07**	6.70	41.88	6.30	39.38

benefits of applying efficient asynchronous schemes to undertake the parallelization of multiobjective metaheuristics for phylogenetics on shared-memory multicore multiprocessor systems.

Concluding Remarks

This first experimental scenario on shared-memory architectures has provided insight into the parallel opportunities shown by each metaheuristic design, as well as the relevant contribution which supposes the introduction of novel nature-inspired asynchronous parallelization schemes. The comparative study of the six examined metaheuristics under synchronous parallelization models has allowed us to identify three main metaheuristic groups, in accordance with the quality of their parallel results. The first group contains MO-FA as the synchronous parallel design which best benefits from the exploitation of shared-memory multicore systems to reduce the execution time required by the metaheuristic. The evolutionary approaches in IBEA, NSGA-II, and SPEA2 also represent suitable designs for parallelization purposes, obtaining the second best parallel results.

On the other hand, this analysis has pointed out that the parallelization of the third group, containing MOABC and IMOBA, represents a challenging question due to their algorithmic features. In order to develop efficient parallelization schemes for this kind of metaheuristics, we have explored the application of asynchronous nature-inspired schemes, using MOABC as case study. Our results have shown the advantages of the asynchronous non-generational model, which is able to minimize the major performance problems shown by the synchronous generational counterpart without losing the multiobjective quality of the inferred solutions. As a result, the asynchronous implementation of MOABC allows the metaheuristic to take full advantage of shared-memory resources, achieving near-optimal speedups while outperforming other parallel approaches from the state-of-the-art. The success of AN-MOABC in tackling in an efficient way a grand computational challenge like phylogenetic inference suggests the major role that such asynchronous models will play in future research efforts on the development of parallel bioinspired algorithms. In fact, a well-defined line of future work lies on the study and modelling of asynchronous behaviour patterns in bats to apply this kind of parallelization schemes to our best metaheuristic in terms of multiobjective quality, IMOBA.

8.3 Evaluation of Mixed Mode Approaches

This section reports the experimental assessment of mixed mode approaches to parallelize our metaheuristics on hybrid shared-distributed memory systems. Firstly, we justify the relevance of mixed mode programming over pure MPI designs, showing the advantages of using OpenMP at the intra-node level. Afterwards, a comparative study between MO-FA and NSGA-II is introduced to examine which metaheuristic leads to a better exploitation of multicore clusters. Finally, the evaluation of parallel performance on constellation-like systems is undertaken by using IBEA as case study.

8.3.1 Justifying the Mixed Mode Parallelization Approach

First of all, we examine the suitability of the mixed mode programming model to parallelize metaheuristics on multicore clusters, discussing why a hybrid approach can lead to performance improvements over a pure MPI code. For this purpose, we have analyzed the intra-node performance obtained

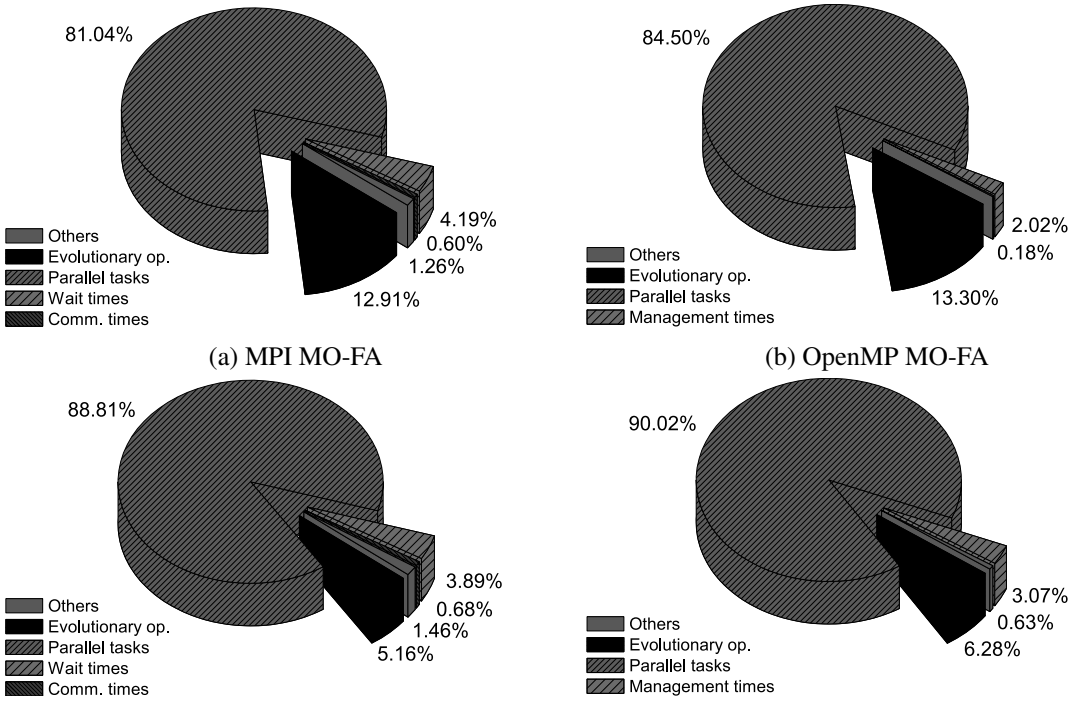
by pure MPI implementations in comparison with pure OpenMP designs, making experiments over a computing node composed of two quad-core Intel Xeon E5410 processors with 8GB RAM. In a first approach, pure MPI implementations of NSGA-II and MO-FA have been developed under a master-worker model. In this sense, worker tasks involve the tree-building, optimization, and evaluation steps for each solution generated by the master, represented by means of *NxN* floating-point matrices. The pure OpenMP designs address the parallelization of these operations by introducing *#pragma omp for* directives. In this first case, we consider that the computation of evolutionary operators is serialized in both implementations.

Figure 8.4 shows the average time percentages spent on parallel tasks, evolutionary operators, other operations, and possible sources of overhead (communication and waiting times in MPI, and thread management in OpenMP, including waiting times at implicit barriers) for these first implementations when using 8 processing cores. It can be observed that these pure MPI versions suffer from noticeable overhead issues in the task assignment and result communication steps even in an intra-node environment. On the other hand, the OpenMP versions are able to spend more time percentages on performing parallel tasks (which rise from 81.04% to 84.5% for MO-FA, and from 88.81% to 90.02% for NSGA-II), reducing overhead issues with regard to the MPI implementations.

These first approaches point out how the serial computation of evolutionary operators contribute a meaningful percentage of execution time (5.16% - 6.28% in NSGA-II, and 12.91% - 13.30% in MO-FA) which can lead to a negative impact in parallel performance. In a second approach, these operations have been included into the tasks to be parallelized in the pure MPI and OpenMP designs. Updated time percentages for these second implementations are given in Figure 8.5. This figure shows that communication and waiting times in the MPI versions increase significantly when evolutionary operators are considered in the worker tasks. This can be explained by the fact that the computation of offspring individuals in NSGA-II and firefly movements in MO-FA requires a data replication strategy in each worker, communicating parent solutions and dominating fireflies. Such data replication issues, which represent a well-known performance pitfall in pure MPI designs, are successfully addressed by using OpenMP, as suggested by the improved percentages obtained when computing parallel tasks (from 87.95% to 95.25% in MO-FA and from 89.94% to 94.90% in NSGA-II). As a result, an effective reduction of 12.88% (MO-FA) and 8.70% (NSGA-II) in execution time is obtained by the OpenMP implementations with regard to the MPI ones. Therefore, this analysis supports the idea of developing mixed mode approaches where MPI processes at the inter-node level rely on OpenMP to exploit intra-node resources, as proposed in the previous chapter.

8.3.2 Mixed Mode Metaheuristic Designs for Multicore Clusters

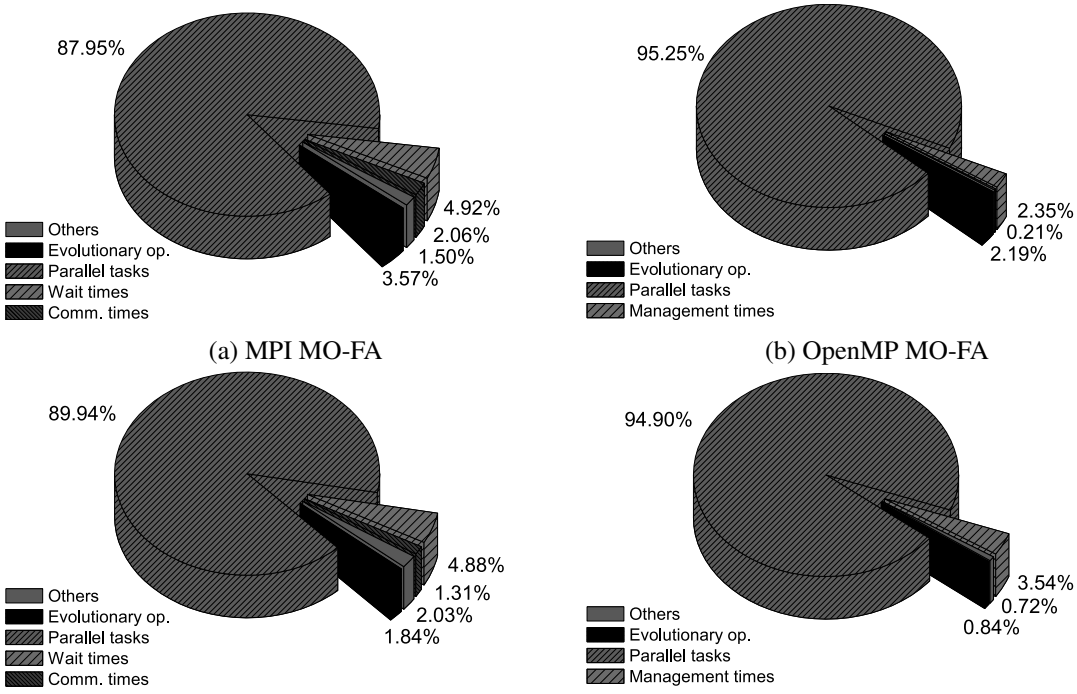
Now we focus on the performance evaluation of mixed mode metaheuristic designs, comparing the parallel results obtained by NSGA-II (representing an evolutionary algorithm with static workload) and MO-FA (representing a swarm intelligence approach with a dynamic number of tasks per generation). For this purpose, we analyze the parallel scalability achieved by these metaheuristics on grow-



(a) MPI MO-FA (b) OpenMP MO-FA

(c) MPI NSGA-II (d) OpenMP NSGA-II

Fig. 8.4 MPI vs. OpenMP - time percentages per operation for the first approach



(a) MPI MO-FA (b) OpenMP MO-FA

(c) MPI NSGA-II (d) OpenMP NSGA-II

Fig. 8.5 MPI vs. OpenMP - time percentages per operation for the second approach

Table 8.15 Hybrid designs for MO-FA and NSGA-II: speedups and efficiencies (static scheduling)

	rbcL_55				*mtDNA_186*				*HIV1_192*			
	MO-FA		NSGA-II		MO-FA		NSGA-II		MO-FA		NSGA-II	
Cores	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)
8	6.722	84.024	**6.851**	**85.643**	6.923	86.539	**7.142**	**89.276**	**7.007**	**87.582**	6.843	85.539
16	12.759	79.748	**13.157**	**82.230**	12.907	80.668	**13.258**	**82.861**	**12.784**	**79.897**	12.708	79.425
32	**23.587**	**73.709**	22.982	71.817	**23.133**	**72.293**	23.107	72.208	21.050	65.780	**22.178**	**69.305**
48	**33.518**	**69.828**	32.040	66.750	**31.875**	**66.407**	29.118	60.662	25.984	54.134	**29.027**	**60.473**
	RDPII_218				*S1482_346*				*ZILLA_500*			
	MO-FA		NSGA-II		MO-FA		NSGA-II		MO-FA		NSGA-II	
Cores	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)
8	**7.081**	**88.509**	6.854	85.675	**7.574**	**94.679**	7.212	90.151	**7.587**	**94.835**	7.252	90.646
16	**12.974**	**81.085**	12.533	78.333	**13.635**	**85.221**	12.958	80.988	**13.756**	**85.976**	13.175	82.346
32	**22.399**	**69.996**	21.942	68.569	**22.682**	**70.880**	22.408	70.024	**22.631**	**70.721**	22.521	70.377
48	**31.729**	**66.102**	28.894	60.196	**30.406**	**63.449**	29.601	61.668	**30.258**	**63.038**	29.947	62.389

ing problem and system sizes, firstly under an OpenMP *static* scheduling policy. For experimental purposes, the following system configurations were considered: 1 MPI process - 8 OpenMP threads (8 cores), 2 processes - 8 threads (16 cores), 4 processes - 8 threads (32 cores), and 6 processes - 8 threads (48 cores). These distributions of threads per process were obtained by experimentation, representing the optimal configurations found for our hybrid designs on this cluster architecture.

In order to evaluate parallel performance, the resulting execution times have been assessed by using the speedup and efficiency metrics, using as reference the median serial times shown in Table 8.3. Table 8.15 details the speedups and efficiencies achieved by MO-FA (columns 2-3, 6-7, and 10-11) and NSGA-II (columns 4-5, 8-9, and 12-13) for the different data sets and system configurations under study. In general terms, this table suggests that MO-FA is able to obtain better parallel results than NSGA-II in most of the problem and system sizes under analysis. More specifically, MO-FA obtains average efficiencies in the range 89.36% (8 cores) - 63.83% (48 cores), improving the ones reported by NSGA-II (87.82% for 8 cores and 62.02% for 48 cores).

These results can be discussed on the basis of the factors which affect the performance of mixed mode parallel algorithms [4]: the parallel efficiency achieved at the worker task processing level (Eff_{worker}), overhead times due to communications (T_{comm}) and synchronizations (T_{sync}), and other processing times related to the serial components of the algorithm and master tasks (T_{master}). Table 8.16 details these factors for the executions which achieved the median speedup values for each metaheuristic on configurations involving 2, 4, and 6 MPI processes (16, 32, and 48 cores).

Firstly, we analyze parallel efficiency at the worker level. This measurement considers the ratio of the time needed by the sequential algorithm to process the compute-intensive operations identified in the profile analysis and the time required by workers to complete their workload. According to Table 8.16, MO-FA outperforms the efficiencies reported by NSGA-II, especially when increasing the number of hardware resources involved in the computations. This can be explained by examining their algorithmic designs. While the number of tasks in NSGA-II remains constant throughout the entire execution of the algorithm, worker tasks in MO-FA evolve dynamically as the algorithm finds new non-dominated Pareto solutions. Under this assumption, our results suggest that the algorithmic

Table 8.16 Analysis of performance factors for the hybrid designs of MO-FA and NSGA-II

	rbcL_55							
	MO-FA				NSGA-II			
Cores	Eff_{worker}	T_{master}	T_{comm}	T_{sync}	Eff_{worker}	T_{master}	T_{comm}	T_{sync}
16	81.489%	7.389s	**1.542s**	**1.132s**	**82.583%**	**3.323s**	1.912s	1.550s
32	**76.008%**	6.956s	**2.708s**	**2.315s**	72.599%	**3.404s**	3.125s	2.914s
48	**72.396%**	7.830s	**2.851s**	**2.795s**	67.376%	**3.811s**	3.555s	20.578s
	mtDNA_186							
16	**83.325%**	68.418s	**17.676s**	**11.881s**	83.137%	**22.340s**	21.374s	15.445s
32	**77.197%**	65.700s	**25.726s**	**14.869s**	72.932%	**23.888s**	30.962s	18.280s
48	**73.153%**	68.457s	**28.759s**	**15.603s**	62.580%	**24.450s**	34.083s	203.966s
	HIV1_192							
16	**83.321%**	46.366s	**9.895s**	12.136s	81.198%	**25.125s**	23.072s	**11.258s**
32	**72.133%**	45.983s	**14.421s**	16.180s	71.875%	**25.559s**	32.250s	**14.537s**
48	**64.748%**	44.901s	**16.611s**	**14.182s**	63.514%	**26.095s**	34.625s	76.636s
	RDPII_218							
16	**83.653%**	57.271s	**14.511s**	**15.410s**	80.312%	**29.625s**	27.715s	20.993s
32	**74.892%**	56.634s	**20.286s**	27.254s	70.683%	**31.494s**	38.028s	**26.779s**
48	**73.371%**	55.353s	**22.477s**	**17.252s**	62.285%	**32.245s**	40.941s	226.384s
	S1482_346							
16	**88.597%**	90.740s	**46.822s**	**14.045s**	84.676%	**72.655s**	67.937s	17.143s
32	**78.047%**	96.716s	**64.542s**	50.876s	74.623%	**77.501s**	95.830s	**18.024s**
48	**74.958%**	97.910s	**68.541s**	**16.599s**	66.702%	**79.268s**	101.553s	217.771s
	ZILLA_500							
16	**90.597%**	215.271s	**74.449s**	**14.351s**	85.984%	**150.177s**	102.882s	18.584s
32	**79.674%**	219.778s	**105.530s**	54.076s	74.983%	**160.566s**	150.036s	**18.686s**
48	**74.802%**	218.718s	**122.491s**	**18.299s**	67.587%	**163.998s**	166.116s	466.345s

design in MO-FA is able to fit accurately the system configuration by itself, giving rise to a good balance of tasks per thread/process and, therefore, improved efficiencies at worker computations. However, it should be noted that *HIV1_192* represents a scenario where MO-FA achieves a weaker efficiency in comparison to other data sets. Due to the characteristics of this alignment (which leads to a high number of solutions in the Pareto front), configurations involving 32 and 48 cores can be affected by the fact that the number of worker tasks to be processed in a generation could be fewer than the number of available cores, giving as a result weaker worker efficiencies (64.75% for 48 cores, in comparison to the average 72.24% shown by the algorithm).

Secondly, communication and synchronization times represent overhead sources that limit the achievable theoretical speedup. Firstly, the communicated messages mostly comprise pseudo-matrix data structures (distance arrays) whose size depends on the number of input sequences. Thereby, higher message delivery times can be observed in both algorithms not only when increasing the number of MPI processes, but also when considering data sets with a high number of species (*S1482_346* and *ZILLA_500*). Secondly, synchronizations among processes can introduce significant waiting times when the system configuration does not allow a balanced distribution of tasks per worker (that is, $tasksNumber\%number\_of\_processes \neq 0$). This issue becomes more noticeable for configurations

Table 8.17 Hybrid designs for MO-FA and NSGA-II: speedups and efficiencies (dynamic scheduling)

	rbcL_55				*mtDNA_186*				*HIV1_192*			
	MO-FA		NSGA-II		MO-FA		NSGA-II		MO-FA		NSGA-II	
Cores	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)
8	6.866	85.825	**6.937**	**86.716**	7.104	88.799	**7.184**	**89.798**	**7.115**	**88.941**	6.989	87.358
16	13.070	81.689	**13.211**	**82.570**	**13.325**	**83.283**	13.311	83.195	**13.150**	**82.189**	12.845	80.283
32	**24.189**	**75.589**	23.147	72.335	**24.249**	**75.779**	23.263	72.697	21.901	68.440	**22.268**	**69.587**
48	**34.216**	**71.283**	32.042	66.755	**33.187**	**69.140**	29.343	61.132	27.234	56.737	**29.152**	**60.733**
	RDPII_218				*S1482_346*				*ZILLA_500*			
	MO-FA		NSGA-II		MO-FA		NSGA-II		MO-FA		NSGA-II	
Cores	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)
8	**7.263**	**90.791**	6.858	85.721	**7.679**	**95.993**	7.215	90.182	**7.745**	**96.817**	7.266	90.824
16	**13.336**	**83.351**	12.652	79.074	**14.012**	**87.574**	13.044	81.525	**14.051**	**87.817**	13.238	82.736
32	**23.260**	**72.688**	22.048	68.901	**22.905**	**71.577**	22.509	70.339	**22.892**	**71.563**	22.641	70.753
48	**32.522**	**67.755**	29.474	61.405	**31.443**	**65.506**	29.847	62.181	**31.218**	**65.036**	30.247	63.014

Table 8.18 Average efficiencies observed when using static and dynamic schedulings

	MO-FA			NSGA-II		
Cores	Static	Dynamic	△Eff	Static	Dynamic	△Eff
8	89.361	91.194	**1.833**	87.822	88.433	0.611
16	82.099	84.317	**2.218**	81.031	81.564	0.533
32	70.563	72.606	**2.043**	70.383	70.769	0.385
48	63.826	65.910	**2.083**	62.023	62.537	0.514

of 6 MPI processes in NSGA-II and 4 processes in MO-FA. The main difference lies on the fact that, while this load imbalance remains as a permanent penalty factor throughout NSGA-II execution due to its static workload features, synchronization penalties are reduced in MO-FA as the metaheuristic evolves towards a number of tasks per process which allows more balanced workload assignments.

Finally, parallel performance is also affected by master tasks and other critical times. As shown in Table 8.16, the computation of evolutionary operators and population management operations in NSGA-II requires less processing time than the firefly movements in MO-FA. The concept of swarm intelligence in MO-FA involves the processing of individuals according to the information gathered by the entire population. As introduced in Chapter 7, the movements of different fireflies (which are dominated by a changing number of individuals) are assigned to different threads, motivating that the attraction inner loops could lead to load imbalances under a static scheduling.

As this issue can be addressed by using more sophisticated loop scheduling policies, we have conducted additional experiments to evaluate NSGA-II and MO-FA under *dynamic* scheduling strategies. The speedup values and efficiencies observed in this second set of experiments are given by Table 8.17. In this context, the proposed scheduling mechanism allows MO-FA to attain a noticeable improvement in efficiency values for all the considered configurations and data sets. In fact, the comparison between static and dynamic mean efficiencies (provided in Table 8.18) shows how MO-FA significantly benefits from a dynamic loop policy, in comparison with the reduced increase in efficiencies obtained by NSGA-II. Moreover, our statistical testing methodology pointed out that MO-FA was able to obtain a statistically significant improvement in speedups over NSGA-II in all

the analyzed data sets when all the 48 cores in our multicore cluster were involved in the computations. Therefore, our results suggest the relevance of MO-FA as the metaheuristic design which attained the best overall exploitation of the considered cluster architecture.

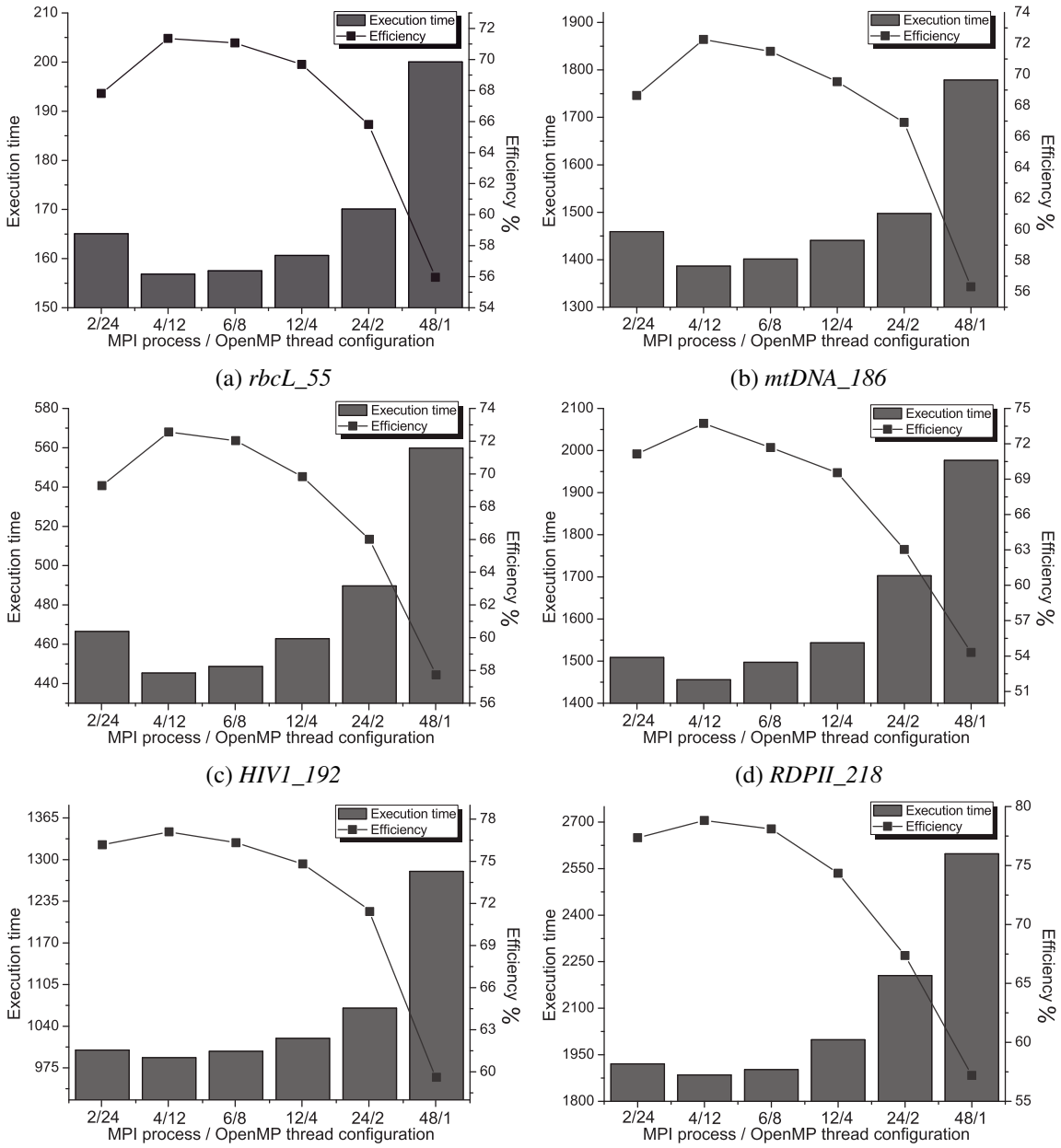
8.3.3 Mixed Mode Metaheuristic Designs for Constellation Configurations

Next, we examine the performance obtained by mixed mode designs (represented by IBEA) on clustered systems in which each computing node contains a high number of processing cores. In this context, it is needed to find the most accurate distribution of MPI processes and OpenMP threads which allows the hybrid design to attain a satisfying exploitation of hardware resources.

As case study, we will focus on examining those configurations which involve the use of all the available processing elements (48 cores). In our experimentation, we have evaluated system configurations composed of 2 MPI processes - 24 OpenMP threads (per process), 4 MPI processes - 12 OpenMP threads, 6 MPI processes - 8 OpenMP threads, 12 MPI processes - 4 OpenMP threads, 24 MPI processes - 2 OpenMP threads, and 48 MPI processes - 1 execution thread (pure MPI implementation). For each dataset, Figure 8.6 represents the parallel execution times achieved by IBEA under the previously described configurations, along with their efficiency values. According to this figure, the best parallel times were obtained when using 4 MPI processes with 12 OpenMP threads per process (that is, one MPI process per processor), while a significant worsening is observed for configurations with a high number of MPI processes.

Our experiments pointed out how the best parallel times were reported by those configurations which achieved a balance between the efficiency at worker computations and the times contributed by master operations and overhead factors. The configuration with the highest number of OpenMP threads (2 MPI processes/24 OpenMP threads) achieved the lowest master times and MPI overhead penalties. However, it also showed the lowest worker efficiency due to the additional times required by OpenMP worksharing directives to manage a high number of threads. On the other hand, the pure MPI implementation (48 processes, 1 execution thread) gave rise to the best efficiencies at worker computations (as OpenMP management times are not involved), while leading to a significant worsening in master times and increased overhead issues. For the remaining configurations, it was verified how growing numbers of MPI processes led to increasing communication and synchronization times due to message passing. The performance penalties contributed by these overhead factors were less significant when using configurations with reduced numbers of MPI processes, taking OpenMP a major role in the computations. As a result, the best parallel times were reported when allocating 1 MPI process per processor, relying in OpenMP to exploit each one of the 12 available cores.

Taking into account these results, we have analyzed the behaviour of the mixed mode implementation of IBEA on different problem and system sizes, evaluating parallel scalability over 8, 12, 24, and 48 processing cores. In this study, we verified that the best configurations of MPI processes/OpenMP threads were in agreement with the previous hints, obtaining the best times when using configurations of 1 process - 8 threads (8 cores), 1 process - 12 threads (12 cores), 2 processes - 12



(a) *rbcL_55* (b) *mtDNA_186*

(c) *HIV1_192* (d) *RDPII_218*

(e) *S1482_346* (f) *ZILLA_500*

Fig. 8.6 IBEA execution times and efficiencies for different MPI+OpenMP configurations (48 cores)

Table 8.19 Hybrid design for IBEA - speedups and efficiencies

	rbcL_55		*mtDNA_186*		*HIV1_192*		*RDPII_218*		*S1482_346*		*ZILLA_500*	
Cores	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)
8	7.030	87.873	7.283	91.042	7.078	88.481	7.291	91.139	7.366	92.073	7.825	97.813
12	10.311	85.928	10.467	87.221	10.452	87.102	10.840	90.330	10.994	91.613	11.579	96.496
24	19.390	80.791	19.528	81.368	19.413	80.887	20.121	83.838	20.378	84.907	21.483	89.513
48	34.262	71.379	34.684	72.259	34.350	71.562	35.267	73.473	37.005	77.095	37.828	78.808

threads (24 cores), and 4 processes - 12 threads (48 cores). Table 8.19 details the median speedups and efficiency values observed with regard to the serial times reported in Table 8.4.

According to these parallel metrics, significant parallel results are achieved by the mixed mode implementation of IBEA in all the considered data sets. Focusing on the results obtained up to 24 cores, the obtained efficiency values are always over 80%, which represents a satisfying exploitation of intra-node multicore resources. When using the entire clustered system (48 cores), relevant speedups are also obtained, in spite of the increasing impact that non-parallelizable times and overhead factors have over parallel performance. In average, IBEA obtains efficiencies of 91.40% (8 cores), 89.78% (12 cores), 83.55% (24 cores), and 74.10% (48 cores). Therefore, we can conclude that the mixed mode implementation of IBEA achieves an accurate exploitation of clustered systems involving a high number of processing cores per node. As a result, the parallel implementation attains significant parallel results, reporting speedups from 34.262 (*rbcL_55*) to 37.828 (*ZILLA_500*) when using the 48 processing cores available in our architecture.

8.3.4 Comparisons with Other Parallel Phylogenetic Methods

In order to assess the relevance of the parallel results obtained by our mixed mode approaches, we undertake comparisons of parallel performance with other hybrid parallel approaches from the literature. In a first step, we will conduct comparisons between MO-FA, RAxML, IQPNNI, and MrBayes over the Intel Xeon E5410-based cluster architecture, proceeding afterwards with the comparative assessment of IBEA and these reference methods over the AMD Opteron 6174-based system.

Table 8.20 provides the comparison of median speedups and efficiencies achieved by MO-FA, RAxML, IQPNNI, and MrBayes. In addition, Figure 8.7 (a) shows the evolution of speedups on increasing numbers of cores for the methods under comparison, considering the mean results obtained in our experimentation. According to these results, MO-FA is able to achieve improved speedup values with regard to these parallel methods in all the studied data sets. In fact, the statistical tests performed over speedup samples reveal that MO-FA obtains a statistically significant improvement over RAxML, IQPNNI, and MrBayes for all the data sets and system configurations. These results can be explained by examining the speedups observed when using 8 cores. These speedups suggest that the hybrid implementation of MO-FA is able to parallelize inferences and evaluations in an efficient way, attaining improved performance in comparison to these well-known hybrid approaches from the literature by making a proper exploitation of multicore resources at the intra-node level.

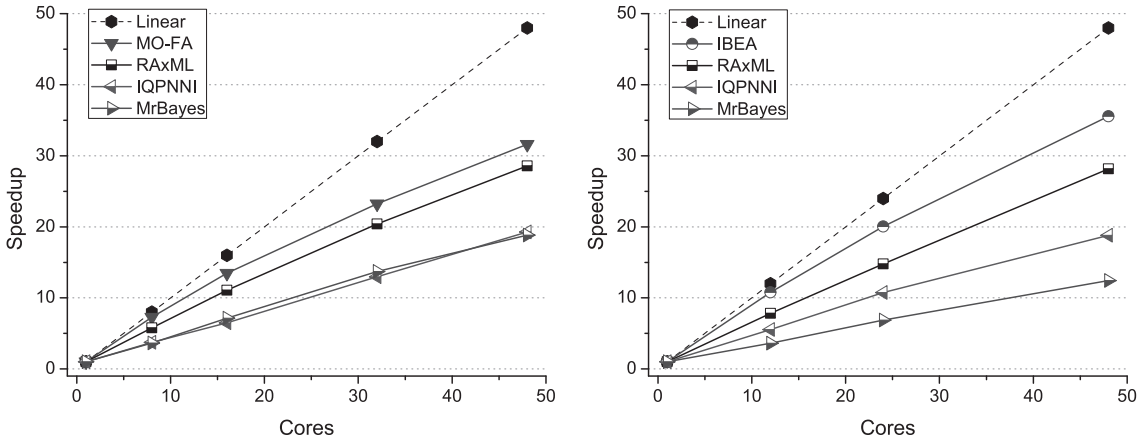
Regarding the results obtained in the second cluster configuration, Table 8.21 gives account of the comparison of median speedups and efficiencies between IBEA and the hybrid biological tools. A graphical representation of the speedups reported by each method on growing numbers of cores is given in Figure 8.7 (b). In overall terms, the mixed mode implementation of IBEA is able to obtain better speedups than RAxML, IQPNNI, and MrBayes on this hardware setup, also achieving a statistically significant improvement in accordance with our statistical testing methodology. Once again, by observing the speedup results reported at the intra-node level (12 and 24 cores), we can

Table 8.20 Comparisons of parallel performance between MO-FA and other parallel methods (MPI+OpenMP hybrid versions)

rbcL_55								
	MO-FA		RAxML-Hybrid		IQPNNI-Hybrid		MrBayes-Hybrid	
Cores	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)
8	**6.866**	**85.825**	5.826	72.825	3.139	39.238	3.306	41.325
16	**13.070**	**81.689**	11.343	70.894	4.451	27.819	6.575	41.094
32	**24.189**	**75.589**	21.802	68.131	8.889	27.778	13.008	40.650
48	**34.216**	**71.283**	31.738	66.121	13.320	27.750	18.220	37.958
mtDNA_186								
8	**7.104**	**88.799**	6.033	75.413	3.819	47.738	5.296	66.200
16	**13.325**	**83.283**	11.857	74.106	6.487	40.544	10.405	65.031
32	**24.249**	**75.779**	22.981	71.816	12.942	40.444	20.154	62.981
48	**33.187**	**69.140**	32.594	67.904	19.322	40.254	28.025	58.385
HIV1_192								
8	**7.115**	**88.941**	5.676	70.950	3.535	44.188	2.775	34.688
16	**13.150**	**82.189**	10.125	63.281	6.236	38.975	5.535	34.594
32	**21.901**	**68.440**	18.112	56.600	12.448	38.900	10.401	32.503
48	**27.234**	**56.737**	25.514	53.154	17.822	37.129	14.435	30.073
RDPII_218								
8	**7.263**	**90.791**	6.202	77.525	5.037	62.963	4.455	55.688
16	**13.336**	**83.351**	11.927	74.544	9.052	56.575	8.811	55.069
32	**23.260**	**72.688**	21.330	66.656	18.061	56.441	17.050	53.281
48	**32.522**	**67.755**	26.675	55.573	27.063	56.381	23.388	48.725
S1482_346								
8	**7.679**	**95.993**	5.572	69.650	2.999	37.488	2.914	36.425
16	**14.012**	**87.574**	10.775	67.344	5.399	33.744	5.530	34.563
32	**22.905**	**71.577**	19.540	61.063	10.770	33.656	10.517	32.866
48	**31.443**	**65.506**	27.516	57.325	15.988	33.308	14.048	29.267
ZILLA_500								
8	**7.745**	**96.817**	5.371	67.138	3.880	48.500	3.110	38.875
16	**14.051**	**87.817**	10.365	64.781	7.385	46.156	5.982	37.388
32	**22.892**	**71.563**	18.610	58.156	14.766	46.144	11.151	34.847
48	**31.218**	**65.036**	27.347	56.973	22.105	46.052	14.983	31.215

verify how our parallel implementation attains a better exploitation of shared-memory resources with regard to these well-known methods for parallel phylogenetic inference. As a consequence, significant speedups are also obtained when using clustered nodes composed of a high number of processing cores, according to the results reported for 48 cores.

Finally, comparisons with the hybrid version of the multiobjective tool PhyloMOEA (using as reference the speedups reported by the authors of this proposal, as described previously in Table 8.14) also support the quality of the results obtained by MO-FA and IBEA. Considering configurations involving 16 cores, we can observe a significant improvement on the four benchmark data sets under study: *rbcL_55* (where MO-FA and IBEA achieve speedups of 13.07 and 13.38, respectively, in comparison with the 8.30 obtained by PhyloMOEA), *mtDNA_186* (13.33 / 13.85 vs. 8.50), *RDPII_218* (13.34 / 14.31 vs. 10.20), and *ZILLA_500* (14.05 / 15.03 vs. 6.30). Therefore, this



(a) Comparisons with MO-FA (b) Comparisons with IBEA

Fig. 8.7 Mean speedups comparisons with other parallel tools on hybrid cluster platforms

Table 8.21 Comparisons of parallel performance between IBEA and other parallel methods (MPI+OpenMP hybrid versions)

rbcL_55								
	IBEA		RAxML-Hybrid		IQPNNI-Hybrid		MrBayes-Hybrid	
Cores	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)	SU	Eff(%)
12	**10.311**	**85.928**	7.125	59.375	4.930	41.083	2.941	24.508
24	**19.390**	**80.791**	13.841	57.671	8.986	37.442	5.701	23.754
48	**34.262**	**71.379**	27.194	56.654	17.723	36.923	10.651	22.190
mtDNA_186								
12	**10.467**	**87.221**	9.215	76.792	5.873	48.942	6.787	56.558
24	**19.528**	**81.368**	16.477	68.654	11.686	48.692	12.852	53.550
48	**34.684**	**72.259**	32.089	66.852	23.087	48.098	24.102	50.213
HIV1_192								
12	**10.452**	**87.102**	7.368	61.400	4.781	39.842	2.449	20.408
24	**19.413**	**80.887**	13.094	54.558	9.414	39.225	4.360	18.167
48	**34.350**	**71.562**	25.612	53.358	17.807	37.098	7.446	15.513
RDPII_218								
12	**10.840**	**90.330**	8.591	71.592	8.386	69.883	4.610	38.417
24	**20.121**	**83.838**	16.986	70.775	16.638	69.325	8.890	37.042
48	**35.267**	**73.473**	32.928	68.600	32.226	67.138	15.089	31.435
S1482_346								
12	**10.994**	**91.613**	7.111	59.258	3.833	31.942	2.412	20.100
24	**20.378**	**84.907**	13.931	58.046	7.463	31.096	4.644	19.350
48	**37.005**	**77.095**	25.516	53.158	13.649	28.435	8.197	17.077
ZILLA_500								
12	**11.579**	**96.496**	7.471	62.258	5.394	44.950	2.559	21.325
24	**21.483**	**89.513**	14.357	59.821	10.329	43.038	4.741	19.754
48	**37.828**	**78.808**	25.542	53.213	20.411	42.523	9.103	18.965

comparative study highlights the significant parallel results obtained by our mixed mode implementations of metaheuristics for phylogenetic inference, attaining relevant results on different cluster configurations in comparison with other state-of-the-art parallel methods.

Concluding Remarks

In this second experimental scenario, we have addressed the parallelization of metaheuristics on hybrid shared-distributed memory cluster systems. A mixed mode programming model represents a suitable approach to take full advantage of this kind of hardware platforms, as OpenMP can be used to address the disadvantages of pure MPI implementations (i.e. population data replications) while accurately exploit shared-memory resources at the intra-node level. In a first case study, we have evaluated the parallel performance achieved by NSGA-II and MO-FA, which represent two dominance-based metaheuristics with well-defined differences in their algorithmic designs. Our experiments on a multicore cluster reveal that the dynamic and adaptive nature of MO-FA is able to accurately fit the characteristics of the underlying hardware, reducing the impact of overhead sources while maximizing efficiency at the worker task processing level.

The second case study considered the evaluation of an indicator-based design, IBEA, on constellation systems in which the interconnected nodes are composed of a high number of processing cores. The optimal exploitation of such platforms is directly related to the way the mixed mode implementation is configured, attending to the number of MPI processes and OpenMP threads per process. For a clustered system composed of two nodes, each one with two twelve-core processors, a configuration which allocated 1 MPI process with 12 OpenMP threads per processor gave rise to the best reduction in execution time when the 48 available cores were involved in IBEA computations.

In conclusion, the parallelization of metaheuristics on hybrid shared-distributed memory environments introduces new questions from a developer perspective. To take full advantage of such systems, the algorithmic design must be able to fit accurately the configuration of the cluster architecture throughout its execution, in order to maximize the exploitation of hardware resources. In addition, the configuration of processes and threads must be performed bearing in mind the characteristics of the system, in order to allow the mixed mode approach to combine the strengths of MPI and OpenMP effectively. As the studied mixed mode designs belong to the synchronous parallelization trend, an open challenge lies on the application of asynchronous hybrid designs to determine if the conclusions reached on shared-memory systems are translatable to multicore cluster platforms.

8.4 Summary

This chapter has undertaken the parallel performance assessment of multiobjective metaheuristics for phylogenetic inference. First of all, we have introduced the two main experimental scenarios considered in this research, detailing experimental conditions and the median times required by the serial versions of the algorithms. These times have served as reference to compute our two measurements of parallel performance: speedup and efficiency. Afterwards, we have reported the results obtained in the first experimental scenario, which was aimed at assessing the parallel performance obtained by our metaheuristics on pure shared-memory systems. In a first step, we focused on evaluating synchronous generational designs for our six metaheuristics, performing experiments on a shared-

memory configuration composed of 24 processing cores. The resulting speedups and efficiencies have given account of the meaningful performance obtained by MO-FA (representing the leading parallel algorithm in the comparisons) and the evolutionary algorithms (IBEA, NSGA-II, and SPEA2). On the other hand, the synchronous approach in IMOBA and MOABC led to weaker speedups in comparison to the remaining methods, as this strategy did not achieve a satisfying solving of the main parallelization challenges included in the algorithmic designs of these two metaheuristics.

In order to deal with the problem of attaining an optimal exploitation of shared-memory resources, we have introduced a comparison between synchronous generational designs and asynchronous non-generational approaches, using MOABC as case study. By examining the speedups and efficiencies observed in our experimentation, we discussed the significant results obtained by the asynchronous approach on a multicore system composed of 48 processing cores (achieving efficiencies in the range 84.78% - 93.25% when using all the cores available in the machine). The temporal analysis of MOABC under synchronous and asynchronous strategies pointed out how the asynchronous approach addressed in a satisfying way the different performance pitfalls shown by the synchronous design, giving as a result an almost optimal exploitation of the shared-memory architecture. Moreover, the comparison with other parallel methods for phylogenetic inference has supported the relevance of the results obtained by the asynchronous approach, outperforming well-known tools such as RAxML, IQPNNI, MrBayes, and PhyloMOEA in terms of parallel scalability.

The second experimental scenario has reported the evaluation of mixed mode implementations for shared-distributed memory hybrid systems. Firstly, we have undertaken a comparative study between the mixed mode versions of MO-FA and NSGA-II on multicore clusters, in order to find out which algorithmic design (static workload vs. dynamic and adaptive workload) was able to fit more accurately the characteristics of a cluster composed of six computing nodes (a total of 48 cores). By analyzing the different performance factors that govern the behaviour of hybrid parallel systems, we found that the mixed mode implementation of MO-FA achieved a statistically significant improvement in parallel performance over NSGA-II, thanks to the improved efficiency observed at the worker computations level and the benefits of applying dynamic loop scheduling policies. Secondly, the challenge of obtaining an optimal configuration of MPI processes/OpenMP threads to take full advantage of clustered systems where each computing node contains a high number of processing cores was addressed. Using for this purpose IBEA on a cluster system composed of two nodes (each one containing two twelve-core processors), our experiments reported improved parallel times when using a configuration which involved the allocation of 1 MPI process per processor, each one composed of 12 OpenMP execution threads. Under this configuration, a significant exploitation of the clustered system was obtained by the hybrid implementation of IBEA, attaining efficiencies over 70% when using all the available resources. The quality of the results obtained by our mixed mode versions of MO-FA and IBEA was confirmed by the comparisons with other hybrid parallel methods from the literature. In conclusion, this chapter has showed the benefits and challenges which suppose the parallelization of metaheuristics for phylogenetics, giving account of how advanced parallel designs are able to accurately tackle this kind of grand computational challenges.

Chapter 9

Heterogeneous Approaches for Phylogenetics

This chapter undertakes the design of heterogeneous CPU+GPU approaches for phylogenetics, in order to explore the benefits of applying parallel schemes at the problem dependent intra-algorithm level. For this purpose, the parallelization of the phylogenetic parsimony function is taken as case study. Starting from the description of Fitch's algorithm in its serial version, GPU and CPU kernels to accelerate parsimony evaluations are proposed, using for this purpose the standard in heterogeneous computing OpenCL. On the basis of these kernel implementations, a CPU+GPU multidevice design which combines CPU and GPU capabilities is introduced, with the aim of maximizing parallel performance when performing complex parsimony computations on heterogeneous platforms.

9.1 Problem Dependent Intra-Algorithm Parallelization and Heterogeneous Computing

As remarked in Chapter 4 - Section 4.2.6, research interests on heterogeneous CPU+GPU systems have acquired a major role in the high performance computing area [53]. By combining the parallel capabilities of multicore CPUs and GPU accelerators, current heterogeneous platforms are able to provide satisfying mechanisms to address the increasing demands of computing power. In this sense, the growing availability of such systems has led to increasing efforts in developing new parallel solutions based on heterogeneous computing. In fact, heterogeneous approaches have shown significant performance when accelerating scientific applications [186], pointing out the relevance of applying heterogeneous CPU+GPU proposals to attain a significant exploitation of the underlying hardware resources [166, 181]. From a bioinspired computing perspective, the fine-grained parallelism capabilities shown by heterogeneous platforms can contribute to the acceleration of optimization procedures, by taking advantage of the data parallelism shown by the evaluation procedures at the objective function loop level.

Computational biology represents one of the main scientific domains that can benefit from the use of parallel heterogeneous systems. Advances in molecular sequencing have led to the generation of increasing amounts of biological data whose processing represents a significant challenge [156]. Moreover, the high computational complexity shown by a wide range of problems in bioinformatics explains the infeasibility of applying serial computational approaches [198]. Fortunately, most biological applications show data parallelism opportunities at the molecular processing level, motivating the growing efforts on developing heterogeneous strategies to minimize execution times [150]. Focusing on the phylogenetic inference problem, the objective functions used in the evaluation procedures often assume that each character in the sequences evolves in an independent way. This assumption opens the door to the design of fine-grained heterogeneous approaches to reduce the linearly-growing evaluation times required to deal with increasing sequence lengths. The literature (as reported in Chapter 3) has given account of the successful parallelization of Bayesian and likelihood-based phylogenetic reconstruction procedures on heterogeneous systems. In this case study, we focus on the maximum parsimony principle, which searches for the simplest evolutionary hypothesis in accordance with the Occam's razor statement [61]. Due to the inherent data parallelism shown by this function, heterogeneous computing represents a promising tool to parallelize parsimony computations on real biological data sets.

In this chapter, the development of CPU and GPU heterogeneous approaches to accelerate the phylogenetic parsimony function is detailed. The main idea consists of exploiting the parallel capabilities of multicore CPUs and GPUs to minimize the execution time required by this well-known function. For implementations purpose, we have used the standard OpenCL [68], a general framework for developing parallel applications on heterogeneous systems. By taking into account the key features and capabilities of each device, CPU and GPU kernels to perform parsimony computations under Fitch's algorithm are proposed. On the basis of these kernels, a CPU+GPU multidevice approach is proposed to parallelize multiple parsimony evaluations, providing a heterogeneous design which aims to take full advantage of the available computing devices to minimize execution time.

9.2 A Case Study on Heterogeneous Computing: Fitch's Algorithm

Phylogenetic methods are built upon the concept of optimality criterion, an objective function which guides the search engine by evaluating the quality of the inferred evolutionary trees. As these evaluation procedures are carried out by making computations over the sequences that characterize the input organisms, increasing sequence lengths give rise to linearly-growing objective function times. On current biological data sets, significant numbers of characters per sequence are comprised and, thereby, the tree evaluation procedures become very computational demanding. As a result, percentages over 80% of the overall execution time are spent on this step [8]. This issue motivates the interest on applying parallel computing to minimize evaluation times in real phylogenetic analyses.

In order to show the benefits of developing heterogeneous approaches to solve this problem, we study the parallelization of the classic maximum parsimony criterion. Parsimony-based methods

give support to the simplest evolutionary history, represented by a phylogenetic topology which minimizes the number of character state changes between ancestor and descendant nodes. Parsimony evaluations are often conducted by applying Fitch's algorithm [61]. In this section, we describe the serial version of this algorithm, identifying its main data dependencies and parallel opportunities, and proposing parallel implementations for CPU and GPU devices.

9.2.1 Description of the Algorithm

Assuming that each character in the input sequences evolves independently, Fitch's algorithm computes the parsimony score of a phylogenetic tree $\tau = (V,E)$ (denoted as $P(\tau)$) by defining an assignment of character state sets S per node that reduces the number of changes observed throughout the topology. As explained in Section 2.2.3 of Chapter 2, this procedure involves two main steps. Firstly, for each character i=1 to M (where M is the length of the sequences), the algorithm proceeds from the leaves to the root, computing for each node the set of possible states S_i as follows. For an OTU l, the state set $S_i(l)$ is given by the value l_i observed at the i-th character of the input sequence corresponding to the organism l. Once the state sets for the leaf nodes have been defined, Fitch's algorithm applies a bottom-up strategy to calculate ancestral state sets. Given a HTU denoted by an internal node u with children v and w, $S_i(u)$ is computed in the following way:

$$S_i(u) = \begin{cases} S_i(v) \cap S_i(w) & \text{if } S_i(v) \cap S_i(w) \neq 0, \\ S_i(v) \cup S_i(w) & \text{if } S_i(v) \cap S_i(w) = 0. \end{cases} \tag{9.1}$$

This expression can be used to identify those evolutionary steps that imply an increase in the parsimony score of the phylogeny. When the intersection $S_i(v) \cap S_i(w)$ is not 0, it means that no substitution event at the i-th character has taken place from the ancestor to the descendants, as the children have inherited the value from the parent. In this case, $P(\tau)$ is not updated. On the other hand, if this intersection is 0, it means that the children do not share any common state value at i. Therefore, a substitution (mutation) event has taken place from the parent to, at least, one of its children. When this situation is detected, the parsimony score $P(\tau)$ increases. Figure 9.1 shows a graphical explanation on how this step proceeds.

The second step in Fitch's algorithm defines the assignment of final state sets for each internal node, starting from the root r. The final root state r_i is set by choosing any state value in $S_i(r)$. Afterwards, each remaining internal node v with parent u is processed, in such a way that v_i takes the value u_i iff $S_i(v)$ contains u_i. Otherwise, any state value in $S_i(v)$ is chosen as v_i.

When applying Fitch's algorithm, two key considerations should be taken into account. First of all, if we are mostly interested in computing the parsimony scores of a phylogenetic topology, the second step of the algorithm can be ignored, as $P(\tau)$ is calculated throughout the execution of the bottom-up step, as remarked in [8]. Secondly, the input alignment can contain certain characters which do not contribute to the parsimony score. For example, if the j-th characters in the input sequences show the same state value for each organism, the sets S_j will always include this state

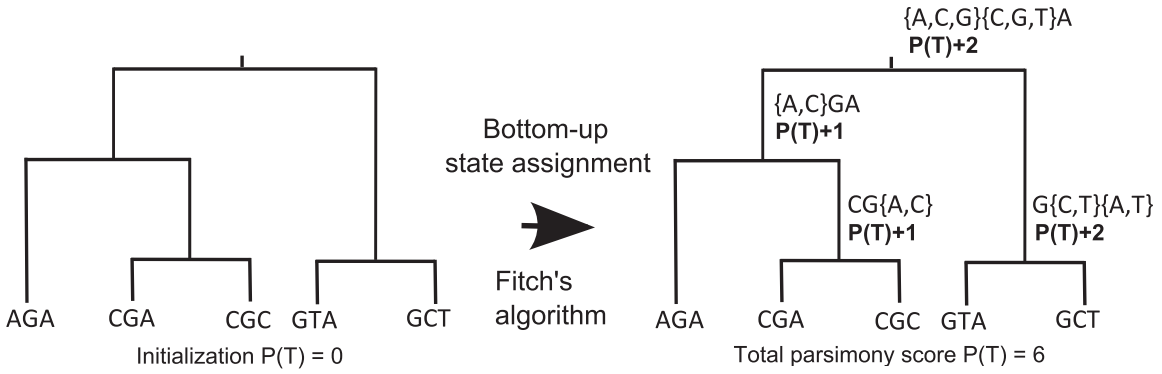


Fig. 9.1 Example of Fitch's algorithm

and, therefore, the parsimony value at the site j will be 0. These characters that do not contribute to the parsimony score are known as parsimony non-informative sites, whose processing can also be ignored [8]. This study addresses the acceleration of Fitch's algorithm for parsimony computations (bottom-up step), considering data sets in which all their characters are parsimony informative sites.

9.2.2 Parallelizing Fitch's Algorithm: General Features

After describing the serial version of Fitch's algorithm, we study now heterogeneous CPU/GPU approaches to parallelize parsimony computations through this approach. For this purpose, we have used the open standard OpenCL. In an OpenCL application, a CPU host prepares and assigns tasks to one or more computing devices containing multiple compute units, whose processing elements execute task instructions. More details on OpenCL programming are available in Chapter 4 - Section 4.2.7 and reference [68]. To achieve significant performance in a heterogeneous context, we need to identify the major sources of parallelism and data dependencies of the problem, along with designing proper kernels to exploit the capacities of each computing device.

To parallelize Fitch's algorithm, several considerations have to be taken into account. Firstly, this algorithm shows data dependencies at the topological processing level. For an internal node u with children v and w, the computation of its state set for a character i $S_i(u)$ depends on $S_i(v)$ and $S_i(w)$. That is, $S_i(u)$ (and the parsimony update) cannot be defined until $S_i(v)$ and $S_i(w)$ have been calculated. On the other hand, this algorithm considers that each character evolves independently, representing the main source of parallelism of the approach. Under this assumption, the calculation of S_i does not show dependencies with regard to the calculation of the sets S_j for any remaining character j. Therefore, the parsimony score can be defined in terms of partial parsimony values $P_i(\tau)$ for each character i=1 to M, whose computations are suitable to be carried out in a parallel fashion.

The parallel design of Fitch's algorithm for parsimony evaluation takes as input a char-type matrix representing the nucleotide sequences under study and a set of *numTrees* phylogenetic trees to be evaluated. In this implementation proposal, the characters in the input sequences are codified by means of hexadecimal values, as described in Table 9.1. This table considers the different states

Table 9.1 Hexadecimal codification of the input sequences

Character	Hexadecimal	Description	Character	Hexadecimal	Description
'A'	0x08	Adenine	'Y'	0x05	Cytosine or Thymine
'C'	0x04	Cytosine	'K'	0x03	Guanine or Thymine
'G'	0x02	Guanine	'V'	0x0E	Adenine or Cytosine or Guanine
'T'	0x01	Thymine	'H'	0x0D	Adenine or Cytosine or Thymine
'M'	0x0C	Adenine or Cytosine	'D'	0x0B	Adenine or Guanine or Thymine
'R'	0x0A	Adenine or Guanine	'B'	0x07	Cytosine or Guanine or Thymine
'W'	0x09	Adenine or Thymine	'?'	0x0F	Unknown or missing data
'S'	0x06	Cytosine or Guanine	'-'	0x10	Gap

that can be found in modern data sets, so that the design is able to define a realistic approach for parsimony computations. Under this codification, efficient bitwise logical operations AND and OR can be applied to perform the intersection and union operations required by the algorithm. For storage purposes, we consider an array of *NxM* characters enclosing the entries of the alignment matrix in row-major order, that is, the first *M* elements in the dataset array contain the first input sequence, the next *M* elements contain the second sequence, and so on.

Regarding strategies for topology representation, the input phylogenetic trees to be evaluated are codified by using the *TreeTemplate* class from Bio++ [75]. From these tree-shaped objects, an optimized data structure specifically designed for its processing at the kernel level is defined. As Fitch's algorithm operates over the internal nodes of the topology, a struct type *cl_node* is specified to codify such nodes. The proposed struct is composed of two fields: a short integer representing the number of children nodes, and an integer array containing children identifiers. In these children identifiers, the most significant bit is used to distinguish between internal nodes (denoted with a value=1) and leaf nodes (0). Therefore, a topology for kernel processing is given by a *cl_node* array of internal nodes, whose elements are organized according to the post-order tree traversal of the input phylogeny to ensure correct data dependencies.

The host side of the application is responsible for managing these data structures, transferring them to the computing devices in order to execute the parsimony kernel. At each iteration, the application conducts the evaluation of one of the *numTrees* input phylogenies. Host tasks involve the initialization of the *cl_node* topology array, the dataset and topology array mapping/transfer from the host memory to the device memory (it is noted that the transfer of the dataset array is only required in the first iteration), the transmission of the kernel execution command via *clEnqueueNDRangeKernel*, and the reading of the calculated parsimony score from the device. These tasks are repeated until all the *numTrees* phylogenetic topologies in the input of the procedure have been evaluated.

Kernel implementations must consider the particular device capabilities to obtain significant speedups. Current multicore CPUs benefit from implementations in which a coarse-grained workload is assigned to each parallel thread, enhancing performance by relying on vectorization strategies supported on SIMD extensions. On the other hand, GPUs show a highly multithreaded architecture and, thereby, the definition of a large number of fine-grained parallel tasks is required to attain a satisfying exploitation of GPU resources. Next, we detail GPU and CPU kernel proposals for parsimony.

9.2.3 GPU Kernel - Implementation Details

Maximizing data parallelism is the key principle behind GPU programming. This basic idea implies that GPU kernels must be designed with the goal of taking full advantage of the data parallelism opportunities shown by the application. Under this consideration, our parsimony kernel implementation for GPU devices is defined as described in Algorithm 29. In this kernel, each work-item with identifier *item_id* will carry out the computation of the partial parsimony score $P_{item\_id}(\tau)$ for the *item_id*-th characters of the input sequences.

Algorithm 29 GPU parsimony kernel

Input: *cl_node* tree* (constant memory), *charn* dataset* (global memory), *int char_per_sequence* (constant), *int num_internal_nodes* (constant), *int* shared_reduction* (local memory).
Output: *int* parsimony_output* (global memory).

```
1:  item_id ← get_global_id(0)
2:  group_id ← get_group_id(0)
3:  P_item_id ← 0
4:  S_item_id ← 0
5:  for i = 1 to num_internal_nodes do
6:      node_state ← 0x1F
7:      for j = 1 to tree[i].num_descendants do
8:          /* Retrieving information from the child j */
9:          node_type ← tree[i].children_id[j] & 0x80000000
10:         node_id ← tree[i].children_id[j] & 0x7FFFFFFF
11:         /* Reading state from the previously computed ones (internal node) or from the dataset (leaf node) */
12:         if node_type = 0x80000000 then
13:             child_state ← S_item_id[node_id]
14:         else
15:             child_state ← dataset[char_per_sequence*node_id+item_id]
16:         end if
17:         /* Computing node state and updating partial parsimony scores */
18:         partial_state ← node_state & child_state
19:         P_item_id,node_state ← (partial_state=0) ? {P_item_id+1, node_state | child_state} : {P_item_id, partial_state}
20:     end for
21:     S_item_id[i] ← node_state
22: end for
23: parsimony_output[group_id] ← Warp-based Parallel Reduction (P_item_id, shared_reduction)
```

For this purpose, the defined work-items process each one of the internal nodes in the order they appear in the topology array, in order to compute the corresponding states S_{item_id}. For each internal node, the information about their children nodes is retrieved (lines 8-10 in Algorithm 29), obtaining their node types and identifiers. The identifier value denotes a position in the topology array for internal nodes, while a leaf identifier refers to the corresponding sequence in the input dataset. Therefore, if the currently processed child represents an internal node, its state will be read from the S_{item_id} array, which contains the states previously computed by the work-item (line 13). Otherwise, the current child is a leaf node, and its character state value will be read from the dataset array (line 15). Once child information has been processed, the state set operations in Fitch's algorithm are performed, updating accordingly the partial parsimony score and the state value of the currently processed node (lines 18-19). These tasks are performed for each child node, storing the resulting state value in the S_{item_id} array. Afterwards, the kernel goes on with the next internal node, repeating the previous steps until all the nodes in the topology array have been processed. In the final

steps of the kernel, the partial parsimony scores calculated by each work-item in a work-group are combined by applying a warp-based parallel reduction [126].

In this implementation, the GPU memory hierarchy is used in the following terms. Firstly, the topology array is mapped to the constant memory, as it represents a read-only data structure. On the other hand, the dataset array is stored at the global memory, due to the fact that it is the largest data structure managed in the algorithm and requires a high amount of memory. The row-major organization adopted in this array avoids the emergence of strided access patterns to the global memory within a warp, as work-items perform row-wise accesses to sequence characters (in such a way that the k-th thread uses the k-th gathered element). In addition, the global memory contains the output parsimony score, which will be transferred from the device to the host upon termination of the kernel execution. Local memory at the work-group level is reserved to conduct parallel reduction operations. Finally, the local variables used by a work-item are stored at the private memory.

Regarding kernel optimization techniques, this implementation considers the use of vector data types (such as *intn*, *charn*), which allow each work-item to carry out multi-character accesses and operations [147]. This design can benefit the GPU implementation in terms of increased bandwidth utilization and reduced instruction count. Regarding work-item behaviour, the if-else condition located at the lines 12-16 does not lead to branch divergence, as all the threads inside a warp will be processing information about the same child node, following the same execution path. Further improvement in execution time can be achieved at the host-device interaction level, by overlapping initializations and memory transfers with kernel computations [125].

9.2.4 CPU Kernel - Implementation Details

Regarding kernel implementations for CPU devices, a proper exploitation of hardware resources can be attained by combining the definition of coarse-grained parallel tasks with the SIMD capabilities in current multicore CPUs. Our CPU parsimony kernel proposal (described in Algorithm 30) defines that each work-item will be responsible of performing a set of M/*num_work-items* parsimony computations. At the i-th iteration, the work-item processes the topology array with the aim of computing S_i and $P_i(\tau)$ for each internal node. For this purpose, children states are retrieved accordingly with their position in the topology (lines 10-18 in Algorithm 30), carrying out Fitch's operations afterwards to calculate the ancestral state and partial parsimony score of the current node (lines 19-21). Once all the children nodes have been processed, the computed state is stored in the S_{item_id} array, going on with the next internal node. After processing the entire topology, the work-item proceeds with the calculations corresponding to the next character.

In order to boost CPU performance, two main optimizations are introduced. In the first place, memory access patterns are improved by reorganizing the dataset array in column-major order. In this way, the i-th characters in the sequences are stored in contiguous positions of this array, being accessed for the processing of leaf states at the i-th iteration (line 17). In the second place, explicit vectorization techniques [88] through the use of vector data types are applied to enhance perfor-

Algorithm 30 CPU parsimony kernel

Input: *cl_node* tree* (constant memory), *charn* dataset* (global memory), *int initial_char* (constant), *int final_char* (constant), *int num_internal_nodes* (constant), *int num_leaf_nodes* (constant).
Output: *int* parsimony_output* (global memory).

```
1:  item_id ← get_global_id(0)
2:  P_item_id ← 0
3:  S_item_id ← 0
4:  for i = initial_char to final_char do
5:      /*Arrangement of the memory data to be accessed in the current iteration*/
6:      partial_seq_data ← dataset + i*num_leaf_nodes
7:      for j = 1 to num_internal_nodes do
8:          node_state ← 0x1F
9:          for k = 1 to tree[j].num_descendants do
10:             /* Retrieving information from the child k */
11:             node_type ← tree[j].children_id[k] & 0x80000000
12:             node_id ← tree[j].children_id[k] & 0x7FFFFFFF
13:             /* Reading state from the previously computed ones (internal node) or from the dataset (leaf node) */
14:             if node_type = 0x80000000 then
15:                 child_state ← S_item_id[node_id]
16:             else
17:                 child_state ← partial_seq_data[node_id]
18:             end if
19:             /* Computing node state and updating partial parsimony scores */
20:             partial_state ← node_state & child_state
21:             P_item_id,node_state ← (partial_state=0) ? {P_item_id+1, node_state | child_state} : {P_item_id, partial_state}
22:         end for
23:         S_item_id[j] ← node_state
24:     end for
25: end for
26: parsimony_output[item_id] ← P_item_id /* Reductions performed at the host side */
```

mance by exploiting SIMD parallelism. This technique allows work-items to carry out multiple computations by using the CPU vector processing units (for state and parsimony calculations), with the additional advantage of enabling wider memory transfers (at the dataset reading level).

9.2.5 Heterogeneous CPU+GPU Multidevice Approach

These kernels lay the foundations of a multidevice design defined to parallelize parsimony evaluations on heterogeneous CPU+GPU systems. Given *numTrees* phylogenies to be evaluated under parsimony, the key idea consists of distributing independent tree evaluations among the available CPU and GPU devices, using OpenCL as a common framework to manage host-device interactions and to perform kernel computations. The implementation of this multidevice approach considers multiple host threads, each one handling tree initializations and data movements with a particular computing device. The devices involved in the computations will receive the required data from the corresponding host thread and perform the processing of the kernel associated to the assigned task.

Due to the heterogeneity shown by the considered devices, an accurate exploitation of hardware resources requires identifying proper workload distributions to achieve load balancing. The distribution of tasks can be conducted by using performance measurements, in such a way that larger workloads are assigned to those devices which show shorter kernel execution time. To find out the most accurate distribution of tasks per device, load balancing techniques based on constant performance modeling have been applied in this research [103]. Given a heterogeneous system composed

of m devices D_1, D_2, ..., D_m with performance indicators P_1, P_2, ..., P_m and n_{total} tasks to be processed, this model aims to find a distribution of tasks per device n_1, n_2, ..., n_m that leads to a balance among their execution times (t_1=t_2=...=t_m). Let R_1, R_2, ..., R_m be the relative performance shown by each device with regard to a reference device (for example, the CPU). Hence, for a device i, we say that R_i=P_i/P_{ref}. The partition of n_i tasks to be assigned to i is determined in the following way:

$$n_i = \frac{R_i}{\sum_{j=1}^{m} R_j} n_{total}, \tag{9.2}$$

$$\text{such that } \sum_{i=1}^{m} n_i = n_{total}. \tag{9.3}$$

We have considered the guidelines from [3, 65] to apply this model to our heterogeneous multidevice approach for load balancing purposes. The model design defines a main loop of *maxIterations* which encloses the multidevice code. Starting from an equal distribution of tasks (independent parsimony evaluations) among devices, execution times t_i are measured and the runtime performance for each device P_i is set as the ratio between the assigned workload n_i and t_i. Attending to these values, the assignment of tasks per device is updated by using Equation 9.2, proceeding with the next iteration of the model. In this way, task distributions are refined by considering the real runtime performance shown by CPU and GPU devices when performing parsimony calculations in a heterogeneous context. The output of the model will be given by the distribution of phylogenetic trees per device n_1, n_2, ..., n_m that minimizes the overall execution time of the application.

The success of this design will rely on combining CPU and GPU capabilities accurately according to the characteristics of the input sequences. The performance of this approach will be examined in the next chapter by reporting experimental results on real DNA alignment data sets.

9.3 Summary

This chapter has detailed the design of heterogeneous strategies to parallelize phylogenetic evaluations on systems composed of CPU+GPU computing devices. The main goal was to provide examples on how current heterogeneous platforms can be useful to support phylogenetic analyses via problem dependent intra-algorithm parallelization schemes. For this purpose, we have addressed the parallelization of the well-known Fitch's algorithm for performing parsimony computations. Initially, the main data dependencies shown by the algorithm at the topological processing level have been identified, as well as the data parallelism opportunities provided at the character processing level. Afterwards, the main features of our heterogeneous implementations have been defined, describing interactions between the CPU host and the computing device in an OpenCL framework, along with defining optimized input structures specifically designed for its kernel processing.

From these common features, GPU and CPU kernel implementations of the phylogenetic parsimony function have been reported. These kernels were implemented taking into account the parallel

capabilities of each computing device. Firstly, a highly multithreaded and fine-grained implementation was proposed for the GPU parsimony kernel, defining a large number of independent parallel tasks (one partial parsimony computation per work-item) and an accurate usage of the GPU memory hierarchy. Secondly, the CPU parsimony kernel was built upon the combination of more coarse-grained tasks (multiple partial parsimony computations per work-item) and a vectorization-oriented approach to exploit SIMD capabilities in multicore CPUs. In order to take full advantage of current CPU+GPU heterogeneous systems, we have defined a heterogeneous multidevice design in which both CPU and GPUs are jointly used to perform independent parsimony evaluations. In this heterogeneous context, performance modelling procedures have been used with the aim of defining an accurate distribution of tasks for each device, allowing the system to reach load balance and improved performance when dealing with complex phylogenetic data sets.

Chapter 10

Performance Assessment of Heterogeneous Approaches

This chapter addresses the performance assessment of heterogeneous CPU+GPU approaches to accelerate parsimony computations under Fitch's algorithm. To this end, two main evaluation scenarios are considered. Firstly, the execution times reported by our CPU and GPU implementations of the phylogenetic parsimony kernel are examined separately, reporting speedup results with regard to the serial version of the algorithm. Secondly, we evaluate multiGPU and heterogeneous CPU+GPU multidevice configurations, identifying a proper assignment of tasks per device in accordance with the characteristics of the input data. Finally, comparisons with other approaches for parsimony are introduced to validate the relevance of our heterogeneous designs.

10.1 Initial Considerations

We summarize in this section the experimental methodology followed to assess our heterogeneous parallel approaches to accelerate the phylogenetic parsimony function. This parallel performance study was carried out by conducting experiments on a range of real biological data sets from the literature. These data sets belong to the second set of alignments defined in Chapter 4 - Section 4.4: *M8631_50*, a dataset of Enterobacteriaceae DNA, containing 50 sequences (with 42456 nucleotides per sequence); *M21119_55*, a dataset of Caecilian Amphibian DNA containing 55 sequences (4675 nucleotides per sequence); *M19259_156*, a dataset of Salmonella enterica DNA, containing 156 sequences (119750 nucleotides per sequence); *M10272_169*, an alignment of mammalian genomic data, containing 169 sequences (24251 nucleotides per sequence); *ZILLA_500*, an alignment of rbcL plastid gene data, containing 500 sequences (759 nucleotides per sequence); and *HIV1_1459*, an alignment of HIV1 nucleotide data containing 1459 sequences (8610 nucleotides per sequence).

In order to ensure the statistical reliability of the obtained results, 31 independent runs of our application were conducted per experiment. An independent execution involves the evaluation under parsimony of *numTrees*=2000 phylogenetic trees. With the aim of avoiding performance issues when

Table 10.1 Reference serial times (in seconds) for Fitch's algorithm

	M8631_50	*M21119_55*	*M19259_156*	*M10272_169*	*ZILLA_500*	*HIV1_1459*
Serial time	46.999	5.951	498.024	99.142	10.274	323.861

prime sequence lengths are considered, the number of nucleotides per sequence in the dataset array was rounded up to fit the number of compute units in the devices used to perform kernel computations. This technique involved the use of parsimony non-informative sites, so that the introduction of these characters do not have an impact on the output of the algorithm (as such sites contribute zero to the parsimony score of the evaluated phylogeny).

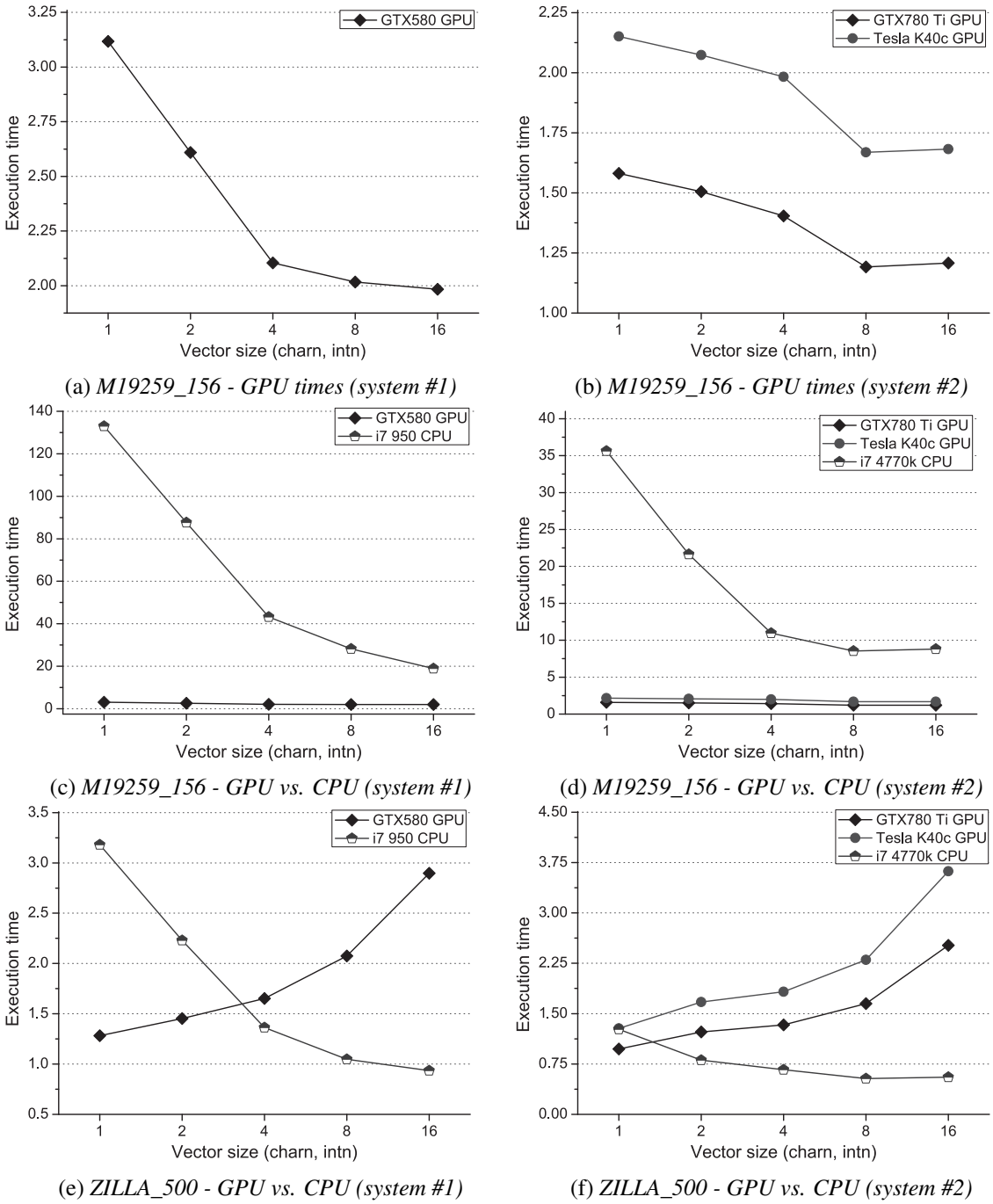
The experimentation which supports this study was conducted on the two different heterogeneous CPU+multiGPU systems with openSUSE OS introduced in Chapter 4 - Section 4.4. The first system comprises an Intel i7 950 CPU at 3.07GHz with 12GB RAM (8 compute units - 4 physical CPU cores with hyperthreading), and two GPUs NVIDIA GTX 580, each one running at a GPU Clock rate of 1.59GHz and 1.54GHz, respectively (16 compute units - streaming multiprocessors and 512 GPU cores). The second platform includes an Intel i7 4770K CPU at 3.5GHz with 32GB RAM (8 compute units - 4 physical CPU cores with hyperthreading), a first GPU NVIDIA GTX 780 Ti running at a GPU Clock rate of 1.05GHz (15 compute units - streaming multiprocessors, 2880 GPU cores), and a second GPU NVIDIA Tesla K40c running at 0.75GHz (15 compute units - streaming multiprocessors, 2880 GPU cores). The tested programs were compiled by using GCC with the -O3 optimization flag enabled and the Intel OpenCL 1.2 and NVIDIA OpenCL 1.1 implementations.

For the sake of uniformity, this study takes as reference the serial execution time obtained on the Intel i7 950 CPU to calculate parallel performance. Hence, the speedups attained by our heterogeneous approaches to accelerate parsimony computations will be reported with regard to these times. Table 10.1 shows the median times per dataset shown by the serial version of the algorithm.

10.2 Single-Device Results

This section analyzes the parallel results observed on single-device configurations. Firstly, we study the effect of performing multi-character parsimony computations in CPU and GPU kernels by means of OpenCL vector data types. Two representative problem sizes are considered for this purpose, given by the data sets *M19259_156* (longest sequence length) and *ZILLA_500* (shortest sequence length). Figure 10.1 shows how execution times evolve on these data sets for each computing device included in the two heterogeneous systems, according to the different vector sizes allowed in OpenCL. For simplicity reasons, we report GPU results from the first GPU in the first system configuration (GTX 580 at 1.59GHz), due to the similarities between the two GPU devices installed in this system.

By examining the results on the *M19259_156* dataset, we can observe how the use of vector data types leads to a significant improvement in execution time for both CPU and GPU implementations (Figures 10.1 (a)/(c) for system #1 and 10.1 (b)/(d) for #2). More specifically, kernel computations



(a) *M19259_156 - GPU times (system #1)*

(b) *M19259_156 - GPU times (system #2)*

(c) *M19259_156 - GPU vs. CPU (system #1)*

(d) *M19259_156 - GPU vs. CPU (system #2)*

(e) *ZILLA_500 - GPU vs. CPU (system #1)*

(f) *ZILLA_500 - GPU vs. CPU (system #2)*

Fig. 10.1 Vectorization results

benefit from configurations involving char16/int16 (on system #1) and char8/int8 (on system #2) vector types. As a result, an average improvement of 27.80% (GPU) and 80.53% (CPU) is observed in execution time with regard to configurations involving scalar data types (simple char/int). In addition, the comparison of CPU and GPU times suggests that *M19259_156* represents a suitable

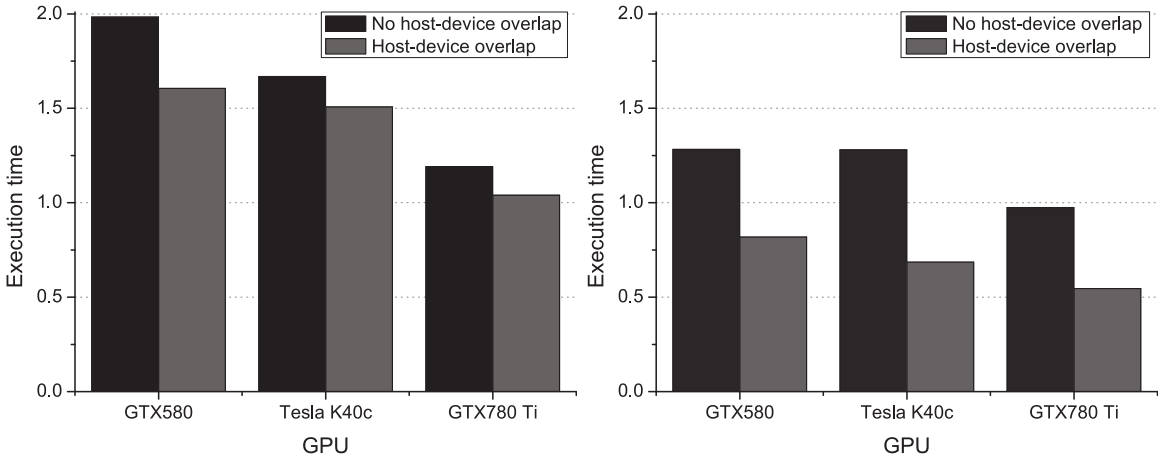
Table 10.2 GPU execution time analysis (in seconds)

System #1					
GPU Device: GTX580					
Dataset	$T_{init+dt}$	$\%_{init+dt}$	T_{kernel}	$\%_{kernel}$	T_{total}
M19259_156	0.386	19.487%	1.597	80.513%	**1.983**
ZILLA_500	0.532	41.516%	0.750	58.484%	**1.282**
System #2					
GPU Device: GTX 780 Ti					
Dataset	$T_{init+dt}$	$\%_{init+dt}$	T_{kernel}	$\%_{kernel}$	T_{total}
M19259_156	0.129	10.873%	1.061	89.127%	**1.190**
ZILLA_500	0.283	29.068%	0.690	70.932%	**0.973**
GPU Device: Tesla K40c					
M19259_156	0.131	7.857%	1.536	92.143%	**1.667**
ZILLA_500	0.286	22.302%	0.994	77.698%	**1.280**

problem size for GPUs, outperforming the CPU implementation of the parsimony kernel by taking advantage of the high data parallelism shown by this dataset at the character processing level.

On the other hand, Figures 10.1 (e) and (f) point out that *ZILLA_500* represents an opposite scenario, where CPU devices take a predominant role over the GPU ones. In detail, the CPU vs. GPU comparison reveals that the CPU implementations start to report shorter execution times than any GPU configuration when using vector sizes involving char8/int8 (system #1) and char2/int2 (system #2) types. While CPU kernels significantly benefit from vectorization (showing an average time reduction of 64.30%), GPU implementations show a worsening in execution time when this technique is applied. At the kernel execution level, the main reason behind this behaviour is given by the low data parallelism offered by this dataset, which motivates the infeasibility to attain a full exploitation of GPU resources. Furthermore, the introduction of vector data types in GPU implementations reduces fine-grained parallelism opportunities for this dataset, giving as a result worse parallel times.

In this context, we have to consider that GPU performance is also affected by the amount of time spent at the host side on initializing topology arrays and transferring data from the host memory to the GPU memory and vice versa. In order to discuss this issue, Table 10.2 introduces an analysis of the execution time reported for *M19259_156* and *ZILLA_500* on the different GPU devices used in this study. In this table, $T_{init+dt}$ (column 2) and T_{kernel} (column 4) denote the times spent on host tasks/data transfers and kernel computations, respectively. The contribution of these times in the overall execution time required by the application (T_{total}, column 6) are given by $\%_{init+dt}$ (column 3) and $\%_{kernel}$ (column 5). By examining the results in Table 10.2, we can realize how the data transfer overhead and initialization times imply a significant percentage in T_{total}, especially for the *ZILLA_500* dataset. As the number of sequences determines the size of the phylogenetic topology, higher initialization and topology transfer times are imposed, contributing an average of 41.52% (system #1) and 25.69% (system #2) in the overall execution time.



(a) *M19259_156* (b) *ZILLA_500*

Fig. 10.2 Overlapping results - GPU execution times

When no overlap strategies are applied, these tasks give rise to significant waiting times for the GPU, which remains idle until the kernel execution command is processed. By overlapping data initializations and transfers (at the host side) with kernel computations (at the GPU device side), the time required for executing the application can be improved. Moreover, if enough resources are available at the GPU level, independent kernel executions may be overlapped (in accordance with the characteristics of the GPU architecture), giving as a result a boost in parallel performance. Figure 10.2 shows graphically the benefits in performance observed when applying this optimization technique to our design: firstly, improvements of 19.08% and 36.17% in execution time on *M19259_156* and *ZILLA_500*, respectively, are verified by the GTX 580 GPU (Fermi architecture); secondly, improvements of 12.71% / 9.70% on *M19259_156* and 43.97% / 46.50% on *ZILLA_500* are achieved by the GTX 780 Ti and Tesla K40c GPUs (Kepler architecture). It is worth to point out that the results obtained in the Kepler GPUs exceed the $\%_{init+dt}$ percentages reported in Table 10.2, suggesting a partial overlap of independent kernel executions on GTX 780 Ti and Tesla K40c.

Considering these optimizations, we now undertake the evaluation of single-device parallel results for the six different data sets under study. Table 10.3 gives account of the median speedup values achieved by each CPU and GPU device separately, taking as reference the times reported by the serial code (Table 10.1). According to our experiments, the GPU implementations lead to significant speedups for the data sets with the longest sequence lengths: *M8631_50*, *M19259_156*, *M10272_169*, and *HIV1_1459*. More specifically, the best times were obtained when using the GTX 780 Ti GPU from the second system configuration. In this sense, the use of GPU devices to accelerate the phylogenetic parsimony function gives rise to very significant improvements on *M19259_156*, resulting in speedups above 310 (GTX 580), 330 (Tesla K40c), and 478 (GTX 780 Ti).

On the other hand, *M21119_55* and *ZILLA_500* represent two data sets with limited sequence lengths in which CPU implementations are able to achieve significant results in comparison to GPU approaches. In fact, the i7 4770K CPU reported the best parallel times (speedup = 19.32) for

Table 10.3 Speedup results for single-device configurations

	System #1		System #2		
Dataset	GTX 580	i7 950	GTX 780 Ti	Tesla K40c	i7 4770K
M8631_50	186.291	21.751	**259.716**	202.145	49.428
M21119_55	23.588	11.271	**58.692**	48.293	27.647
M19259_156	310.322	26.346	**478.718**	330.490	56.545
M10272_169	201.074	26.109	**269.091**	202.666	54.007
ZILLA_500	12.555	10.979	18.814	14.997	**19.321**
HIV1_1459	121.329	25.395	**193.184**	143.326	47.182

ZILLA_500. These results confirm how sequence length (in terms of number of nucleotides) represents a key feature to be considered when exploiting GPU/CPU capabilities. Such divergence in parallel performance attending to the characteristics of the input alignment justifies the design of heterogeneous CPU+GPU approaches for accelerating parsimony evaluations in real-world scenarios.

10.3 Multidevice Evaluation

After discussing the results obtained in single-device scenarios, this section examines the performance achieved on system configurations involving multiple computing devices, each one managed by a different host thread. Particularly, we have considered multiGPU and CPU+multiGPU approaches designed to exploit the capabilities of heterogeneous systems by distributing independent tree evaluations among devices. In our experimentation, we have applied the performance modelling procedure described in the previous chapter (setting as stop criterion *maxIterations*=11) in order to find a accurate distribution of tasks among devices. According to the workload assignments identified by the model, 31 independent executions per dataset were performed in our experiments to obtain statistically reliable samples of the execution times reported by each heterogeneous configuration.

10.3.1 MultiGPU Results

Firstly, we examine the parallel performance obtained when using multiGPU configurations. For each dataset and system, Table 10.4 shows median speedup values (columns 2 and 5) and the number of phylogenetic trees assigned to each GPU according to the performance modelling (columns 3-4 and 6-7). By examining the distribution of topologies assigned per GPU, it can be observed that the model succeeds in identifying performance disparity between devices. For the first system configuration, the differences reported in GPU clock rate lead to different workloads for the same architecture. In average, 1012 phylogenetic trees are assigned for evaluation purposes to the GTX 580 GPU at 1.59 GHz, while 988 phylogenies are processed by the GTX 580 at 1.54 GHz. As a result, the mean speedups on this system rise from 142.53 (single-device configuration) to 270.07.

Regarding the second system configuration, more noticeable differences in task distributions are even verified, according to the diversity in parallel performance shown by the GPU devices GTX 780

Table 10.4 Speedup results for multiGPU configurations

	System #1 GPU1: GTX 580 at 1.59GHz GPU2: GTX 580 at 1.54GHz			System #2 GPU1: GTX 780 Ti GPU2: Tesla K40c		
Dataset	Speedup	Workload GPU1	Workload GPU2	Speedup	Workload GPU1	Workload GPU2
M8631_50	365.496	1012 trees	988 trees	436.578	1172 trees	828 trees
M21119_55	46.807	1004 trees	996 trees	103.862	1126 trees	874 trees
M19259_156	593.793	1015 trees	985 trees	769.428	1194 trees	806 trees
M10272_169	385.676	1015 trees	985 trees	437.635	1176 trees	824 trees
ZILLA_500	24.245	1014 trees	986 trees	32.410	1149 trees	851 trees
HIV1_1459	204.428	1015 trees	985 trees	329.335	1154 trees	846 trees

Ti and Tesla K40c when accelerating the parsimony kernel. The improved parallel results obtained by the GTX 780 Ti give rise to an average assignment of 1162 phylogenetic trees to be evaluated on this GPU device, in comparison to the 838 phylogenies assigned to the Tesla K40c GPU. For this system, an average speedup of 351.54 is reported by the multiGPU design. The benefits of combining multiple GPU devices to process data sets with increasing sequence lengths are confirmed by the results on *M8631_50*, *M19259_156*, and *M10272_169*. For example, the use of multiGPU approaches in this context reduces the execution time for *M19259_156* from a serial reference of 498.02 seconds to 0.84 (system #1) and 0.65 (system #2) seconds.

10.3.2 Heterogeneous CPU+GPU Multidevice Results

After assessing the parallel results obtained in multiGPU configurations, we proceed now with the evaluation of heterogeneous CPU+GPU multidevice designs. For implementation purposes, OpenCL device fission techniques [89] have been applied to avoid performance issues when a multicore CPU is simultaneously processing multiple host tasks along with kernel computations. In this way, five compute units in the CPU device are reserved for kernel executions, while the remaining three are used to allocate host threads (one thread per device).

Table 10.5 presents the parallel results obtained when using all the computing devices available in our heterogeneous platforms (CPU+multiGPU). In this table, columns 2 and 6 refer to the median speedups achieved in our experimentation. The distribution of trees to the CPU and GPU devices as suggested by the performance model are given by the columns 3-5 (system #1) and 7-9 (system #2). According to these results, the integration of CPU computations into the design improves parallel performance for all the data sets under study. Considering the results obtained in both system configurations, Figure 10.3 shows the average improvement in speedups for each dataset.

We can highlight the major role that CPU resources take when analyzing the data sets *M21119_55* and *ZILLA_500*, relying the evaluation of a significant percentage of input phylogenies to the CPU. More specifically, 348 phylogenetic topologies for *M21119_55* and 600 for *ZILLA_500* are assigned

Table 10.5 Speedup results for heterogeneous CPU+multiGPU configurations

	System #1 - CPU: i7 950 GPU1: GTX 580 at 1.59GHz GPU2: GTX 580 at 1.54GHz				System #2 - CPU: i7 4770K GPU1: GTX 780 Ti GPU2: Tesla K40c			
Dataset	Speedup	Workload CPU	Workload GPU1	Workload GPU2	Speedup	Workload CPU	Workload GPU1	Workload GPU2
M8631_50	372.497	31 trees	990 trees	979 trees	454.913	105 trees	1098 trees	797 trees
M21119_55	50.087	348 trees	828 trees	824 trees	116.898	374 trees	929 trees	697 trees
M19259_156	615.335	45 trees	992 trees	963 trees	800.878	80 trees	1146 trees	774 trees
M10272_169	398.219	58 trees	984 trees	958 trees	457.272	100 trees	1115 trees	785 trees
ZILLA_500	35.050	600 trees	710 trees	690 trees	49.888	756 trees	695 trees	549 trees
HIV1_1459	222.410	142 trees	943 trees	915 trees	351.588	144 trees	1071 trees	785 trees

Table 10.6 Statistical comparison of multiGPU vs. CPU+multiGPU speedups

	System #1	System #2
Dataset	Statistical testing output	Statistical testing output
M8631_50	Stat. Sign. differences (P-value = 4.910E-6)	Stat. Sign. differences (P-value = 1.472E-11)
M21119_55	Stat. Sign. differences (P-value = 1.082E-10)	Stat. Sign. differences (P-value = 1.334E-11)
M19259_156	Stat. Sign. differences (P-value = 1.335E-11)	Stat. Sign. differences (P-value = 1.335E-11)
M10272_169	Stat. Sign. differences (P-value = 1.335E-11)	Stat. Sign. differences (P-value = 1.335E-11)
ZILLA_500	Stat. Sign. differences (P-value = 1.335E-11)	Stat. Sign. differences (P-value = 1.324E-11)
HIV1_1459	Stat. Sign. differences (P-value = 1.334E-11)	Stat. Sign. differences (P-value = 1.335E-11)

to the i7 950 CPU in the first system configuration. Regarding the second heterogeneous system, it is remarkable the distribution of trees per device on *ZILLA_500*, being verified the improved performance attained by the i7 4770K CPU. As a result, a total of 756 phylogenetic trees are assigned to the CPU device for this dataset, in comparison with the 695 and 549 trees which are processed by the GPU accelerators. For these two data sets (*M21119_55* and *ZILLA_500*), final speedup results of 50.09 / 35.05 (system #1) and 116.90 / 49.89 (system #2) are reported by the application.

For the remaining instances, the multidevice design also gives rise to a noticeable improvement in parallel results. Considering the dataset with the longest sequence length (*M19259_156*), speedups over 615 (system #1) and 800 (system #2) can be achieved by taking full advantage of heterogeneous CPU+GPU resources. In average, the speedup of the heterogeneous proposal (considering all the data sets) grows up to 282.27 on system #1 and 371.91 on system #2. In order to provide statistical support to this study, we have compared the speedup samples reported on multiGPU and CPU+multiGPU configurations by using the statistical methodology described in Chapter 4 - Section 4.5. Table 10.6 summarizes the results of this testing, suggesting that the CPU+multiGPU design brings about a statistically significant improvement over multiGPU configurations in all the considered scenarios. Therefore, these results justify the relevance of the CPU+GPU multidevice design, in which the different computing devices comprised in current heterogeneous platforms are used to accelerate time-consuming evaluation procedures in real phylogenetic analyses.

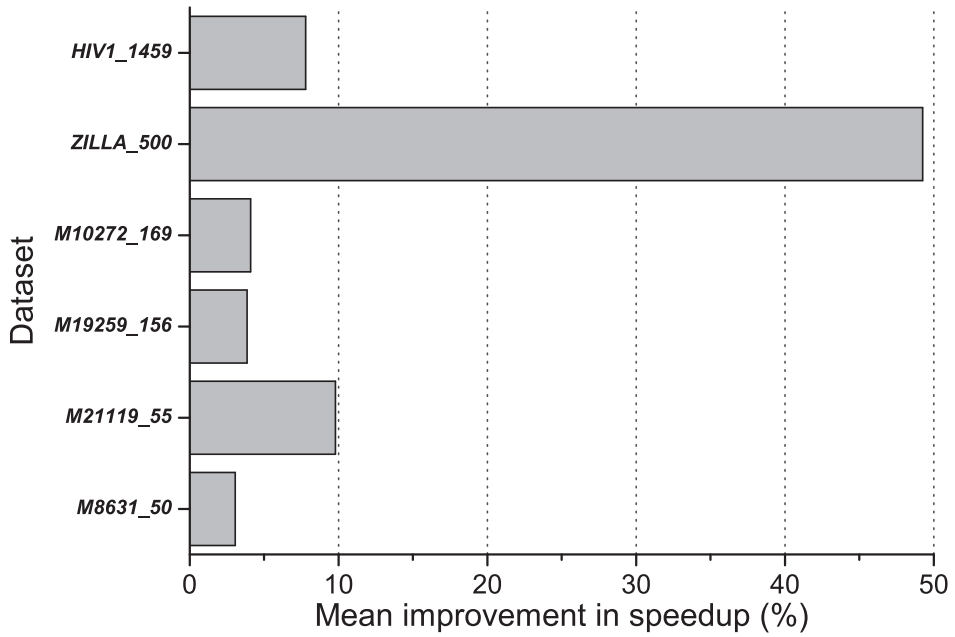


Fig. 10.3 CPU+multiGPU - speedup improvement over multiGPU configurations

Verifying Parsimony Results

We study next the correctness of the parsimony values reported by the implemented kernel. For this purpose, comparisons have been conducted with the parsimony function implementation provided in BIO++, evaluating two sample phylogenetic trees for each dataset. Table 10.7 shows the parsimony scores reported by our approach (columns 2 and 3) and the reference method (columns 4 and 5). As can be observed in this table, the heterogeneous approach returns the correct output in all the considered scenarios, matching the parsimony values calculated by BIO++. Therefore, these results confirm that the parallelization techniques introduced in our kernel proposal do not make an impact on the correctness of the algorithm, preserving the accuracy of parsimony computations while reducing execution time.

10.4 Comparisons with Other Proposals for Maximum Parsimony

Now, we compare our heterogeneous approaches with other proposals from the state-of-the-art. For this purpose, we conduct comparisons of parallel performance with Parsimonator [5], which is considered the fastest open-source implementation of the phylogenetic parsimony function. Parsimonator takes advantage of OpenMP [29] and SSE3/AVX SIMD instructions to accelerate the evaluation of phylogenetic topologies under parsimony. In order to carry out the comparison, experiments with Parsimonator were carried out on the two system configurations considered in this case study. Due to the fact that the OpenMP release of this software requires the use of Intel compilers, we have compiled this software by using *icc* 13. Under these experimental conditions, 31 independent runs

Table 10.7 Comparison of parsimony values returned by the implemented kernel and BIO++

	Heterogeneous Approach		BIO++	
Dataset	Phylogeny #1	Phylogeny #2	Phylogeny #1	Phylogeny #2
M8631_50	371676	374606	371676	374606
M21119_55	38607	38616	38607	38616
M19259_156	430731	446202	430731	446202
M10272_169	274346	274403	274346	274403
ZILLA_500	16474	16595	16474	16595
HIV1_1459	720939	721016	720939	721016

per dataset were performed, each run considering the evaluation of 2000 phylogenetic topologies by using 8 OpenMP threads and enabling SIMD extensions.

Table 10.8 reports the comparison between Parsimonator and our heterogeneous approaches for parsimony. For each single/multidevice implementation, we provide the speedups observed over the OpenMP+SSE3/AVX versions of Parsimonator (named as SU-SSE3 and SU-AVX). More specifically, we compare CPU kernel times between our proposal and Parsimonator, while host/device overlapped execution times are considered for our GPU-based designs. Due to the unavailability of AVX extensions on the Intel i7 950 CPU, speedup results for the first system configuration are only provided with regard to the OpenMP+SSE3 version of Parsimonator.

Focusing on single-device results, our CPU implementation improves the kernel times reported by Parsimonator on *M21119_55* and *ZILLA_500*, while obtaining comparable results on *HIV1_1459*. For data sets with longer sequence lengths, it can be observed how Parsimonator achieves significant performance, as this tool can carry out simultaneous multi-character computations up to 32 nucleotides (in comparison to the 16-length vector types allowed in OpenCL). Parsimonator accomplishes this by treating gaps as unknown characters (same codification for '-' and '?'). Although such approach may improve parallel results, it can also affect the precision of parsimony computations. For example, while our heterogeneous proposal successfully reports scores of 371676 / 374606 for the phylogenies on *M8631_50* (Table 10.7), Parsimonator returns scores of 367411 / 370309 for the same trees. As pointed out in [128], coding gaps as separate characters leads to better biological accuracy and, therefore, is preferable to treating gaps as unknown characters. This is the reason why we decided to codify gaps as specific sites in our implementation.

On the other hand, when GPU accelerators are involved in the computations, our proposal outperforms Parsimonator in all the considered scenarios. As an illustration, noticeable speedups of 5.98 (system #1) and 5.94 / 5.67 (system #2) are observed for the dataset with the longest sequence length (*M19259_156*). These results suggest the relevance of developing GPU-based implementations of the parsimony function, using GPU capabilities to exploit data parallelism opportunities in real biological alignments. Moreover, when using all the computing devices available in the heterogeneous system, improved parallel times can be achieved by distributing independent evaluations among GPU and CPU resources. Considering the results from both system configurations, the proposed multide-

Table 10.8 Parsimonator results and relative speedup achieved by our heterogeneous approaches

	M8631_50	*M21119_55*	*M19259_156*	*M10272_169*	*ZILLA_500*	*HIV1_1459*
System #1						
Parsimonator kernel times (seconds)						
Pars. SSE3	0.938	0.501	9.592	2.424	3.319	9.750
Speedup - Single CPU (i7 950)						
SU-SSE3	0.480	**1.124**	0.513	0.664	**5.706**	0.827
Speedup - Single GPU (GTX 580)						
SU-SSE3	**3.720**	**1.984**	**5.977**	**4.916**	**4.056**	**3.653**
Speedup - MultiGPU						
SU-SSE3	**7.298**	**3.937**	**11.425**	**9.419**	**7.831**	**6.156**
Speedup - CPU+multiGPU						
SU-SSE3	**7.438**	**4.212**	**11.852**	**9.736**	**11.323**	**6.696**
System #2						
Parsimonator kernel times (seconds)						
Pars. SSE3	0.485	0.212	6.179	1.487	1.722	6.475
Pars. AVX	0.438	0.199	5.902	1.412	1.655	6.157
Speedup - Single CPU (i7 4770k)						
SU-SSE3	0.538	**1.277**	0.713	0.869	**6.481**	**1.062**
SU-AVX	0.487	**1.200**	0.681	0.825	**6.227**	**1.010**
Speedup - Single GPU (GTX 780 Ti)						
SU-SSE3	**2.680**	**2.090**	**5.940**	**4.037**	**3.154**	**3.862**
SU-AVX	**2.423**	**1.964**	**5.673**	**3.832**	**3.030**	**3.673**
Speedup - MultiGPU						
SU-SSE3	**4.505**	**3.698**	**9.547**	**6.565**	**5.432**	**6.585**
SU-AVX	**4.073**	**3.475**	**9.118**	**6.233**	**5.220**	**6.261**
Speedup - CPU+multiGPU						
SU-SSE3	**4.694**	**4.162**	**9.937**	**6.859**	**8.362**	**7.030**
SU-AVX	**4.244**	**3.911**	**9.491**	**6.513**	**8.034**	**6.685**

vice design leads to speedups of 11.85 / 9.94 and 9.49 on *M19259_156* over the OpenMP+SSE3 and OpenMP+AVX versions of Parsimonator, respectively. *ZILLA_500* also represents an illustrative scenario where the proposed heterogeneous multidevice design gives rise to relevant parallel performance, improving 11.32 / 8.36 and 8.03 times the results reported by Parsimonator.

Concluding Remarks

This research was carried out with the aim of exploring the implications of applying heterogeneous approaches to accelerate evaluation procedures in complex optimization scenarios. Using as case study the parallelization of the phylogenetic parsimony function, we have shown how current heterogeneous platforms are able to provide high computing capabilities to accelerate the solving of computationally demanding problems like the one tackled in this Thesis. Moreover, we have verified how the nature of the input data plays a major role in the way the underlying computing devices must be exploited. With the increasing availability of sequence data, GPUs may become the preferred

choice to carry out phylogenetic analyses on data sets involving large sequence lengths. However, it should be noted that multicore CPUs can also contribute in a significant way to the acceleration of the evaluation procedures, and even outperforming GPU results in real scenarios like *ZILLA_500*.

Therefore, a heterogeneous CPU+GPU design represents a suitable approach to deal with the challenge that implies the complexity of current biological data sets. According to our experimental results, our proposal successfully addresses the acceleration of parsimony evaluations, achieving significant speedups over the serial algorithm and other parallel approaches proposed in the literature. By means of heterogeneous computing, phylogenetic analyses can benefit from reduced evaluation times, addressing one of the main sources of complexity shown by this problem.

10.5 Summary

This chapter has reported the performance assessment of our heterogeneous approaches to accelerate the phylogenetic parsimony function on current CPU+GPU systems. In order to evaluate the parallel results achieved by the proposal, we have carried out experiments over six real nucleotide data sets on two different heterogeneous systems. Starting from the assessment of multi-character evaluations and host/device overlapping strategies, we have given account of the significant speedups achieved by the proposed kernels. Furthermore, our experiments have pointed out the disparity in parallel performance shown by CPU and GPU devices according to the characteristics of the input data.

Afterwards, the evaluation of heterogeneous multidevice configurations has been undertaken. By using performance measurements to identify accurate workload assignments per device, the proposed multidevice design was able to achieve significant parallel results, showing speedups up to 615x/800x over the serial implementation. In addition, the combination of CPU+GPU capabilities in the multidevice design gives rise to a statistically significant improvement in speedups over multiGPU configurations in all the scenarios under study. Finally, comparisons with other parallel proposals from the state-of-the-art have confirmed the relevance of applying heterogeneous CPU+GPU approaches to accelerate phylogeny evaluations. As a result, the relevance of heterogeneous computing as way to support problem dependent intra-algorithm parallelizations has been verified.

Chapter 11

Conclusions

This chapter details the main conclusions reached as a result of the research undertaken in this PhD Thesis. More specifically, we highlight the most significant findings obtained attending to multiobjective, biological, and parallel perspectives. In addition, new research directions are defined in order to make further progress in future work lines.

11.1 Research Conclusions

This Thesis has studied the application of multiobjective bioinspired metaheuristics and parallel computing to tackle a key problem in the field of bioinformatics: phylogenetic inference. The idea behind phylogenetic reconstruction is to describe hypotheses about the evolution of species by observing the divergence and similarities in their genetic features. In this way, ancestor-descendant relationships can be inferred from an input alignment containing biological sequences from the organisms under study, shedding light on the speciation events that ruled their evolutionary history.

In order to conduct this kind of biological analyses, the definition of an optimality criterion, such as parsimony or likelihood, is often required. The idea is to provide phylogenetic search engines with some guidance mechanisms to distinguish the quality of the inferred phylogenies according to the chosen biological principle. For example, while parsimony gives preference to the simplest evolutionary hypothesis under Occam's razor assumption, likelihood methods rely on probabilistic models to obtain the most likely phylogeny attending to statistical measurements. In this way, phylogenetic engines are able to evolve throughout the execution of the method, moving towards the inference of optimal solutions. However, the different nature of the biological principles supported by these optimality criteria usually leads to the inference of conflicting evolutionary hypotheses, a key issue from a biological perspective. The need to address this major problem has motivated the formulation of phylogenetic inference as a MOP, which seeks to provide not a single solution to the problem, but a set of Pareto solutions which aim to solve such incongruence issues by optimizing two or more criteria simultaneously. In this Thesis, we have focused on solving phylogenetic inference as a bi-dimensional multiobjective problem (according to parsimony and likelihood).

When undertaking the development of computational approaches to phylogenetics, we have to bear in mind the fact that phylogenetic inference is considered one of the most challenging NP-hard problems in bioinformatics. This complexity is motivated by the exponentially growing search spaces that must be processed, along with the time-consuming evaluation procedures which involve the computation of objective functions on current biological data sets. In order to provide effective and efficient tools to deal with this problem, this Thesis has proposed the design and adaptation of different algorithmic approaches which combine bioinspired computing and parallelism.

Conclusions on the Application of Multiobjective Bioinspired Computing to Phylogenetics

Focusing on bioinspired computing, two main trends of multiobjective algorithmic designs have been analyzed throughout the development of this Thesis. Firstly, we have initially tackled the problem by using dominance-based multiobjective metaheuristics, whose fitness assignment procedures are built upon the concept of Pareto dominance. Four dominance-based metaheuristics have been applied in this research: MOABC, a swarm intelligence algorithm inspired by the behaviour of honey bees in nature; MO-FA, a swarm-based design inspired by the bioluminescence capabilities of fireflies; and NSGA-II and SPEA2, two reference evolutionary algorithms which are widely-used in multiobjective optimization. The second main line of multiobjective algorithmic designs studied in this research are the increasingly popular indicator-based approaches, which integrate the computation of multiobjective performance metrics (known as quality indicators) into multiobjective searches for fitness assignment purposes. Representing this second line of algorithmic developments, two indicator-based algorithms have been applied: IMOBA, a swarm intelligence proposal based on the echolocation properties and hunting strategies of bats; and IBEA, one of the most well-known indicator-based evolutionary approaches. In summary, we have proposed three swarm intelligence algorithms (IMOBA, MOABC, and MO-FA) and adapted three reference MOEAs (IBEA, NSGA-II, and SPEA2), summing up to six multiobjective bioinspired metaheuristics to solve the problem.

In order to evaluate the multiobjective and biological quality of the results reported by these metaheuristics, we conducted experimentation over six real biological data sets. The data represented in these alignments involved nucleotide sequences from different species, such as green plants, human mitochondria, HIV1, and prokaryotic organisms. Focusing on the assessment of multiobjetive performance, we performed a comparative study among the six considered metaheuristics under three multiobjective metrics: hypervolume, set coverage, and spacing. In addition, a statistical testing methodology was introduced to provide statistical robustness to the study, allowing us to carry out reliable pairwise comparisons of multiobjective performance.

Under this experimental context, we firstly examined the Pareto fronts reported by each trend of multiobjective optimizers separately. For the dominance-based side, the three multiobjective metrics agreed on pointing out MOABC as the best dominance-based metaheuristic, achieving statistically significant improvements over MO-FA, NSGA-II, and SPEA2 in most of the considered scenarios. MO-FA also reported relevant results with regard to the reference MOEAs, representing the sec-

ond best dominance-based metaheuristic attending to hypervolume and set coverage. These results gave account of how these swarm intelligence designs go a step further over traditional evolutionary approaches in the search for high-quality Pareto solutions to the tackled problem. In this kind of algorithmic designs, the metaheuristic is built upon the definition of different search mechanisms which take into account the knowledge obtained by the entire population (that is, the collective intelligence of the swarm). As a consequence, improved search capabilities are achieved by the swarm, leading to a better processing of the solution space and, thereby, better solution quality. This statement was confirmed in the evaluation of the second trend of algorithmic developments, the indicator-based ones. The swarm intelligence design in IMOBA was able to outperform IBEA from a multiobjective quality perspective, reporting a statistically significant improvement according to our tests.

After assessing each trend separately, we proceeded with the global comparison of dominance-based and indicator-based approaches. The experimental evaluation of these algorithms showed the improved search guidance obtained when integrating quality indicators into metaheuristics. While IBEA was able to report significant results in comparison to the dominance-based MOEAs, NSGA-II and SPEA2, IMOBA succeeded in achieving a statistically significant improvement in multiobjective quality over the dominance-based swarm intelligence designs, MOABC and MO-FA. In fact, IMOBA arose in the overall comparison as our best multiobjective metaheuristic, suggesting that the combination of quality indicator-based searches and swarm intelligence techniques represents a reliable approach to tackle this kind of complex MOPs. This idea was confirmed by the evaluation of IMOBA on the CEC09 benchmarks, which verified the success of the proposed algorithm in obtaining high-quality approximations to the Pareto-optimal fronts for each test instance.

In summary, the experimental evaluation of multiobjective results suggested two key conclusions from a bioinspired computing perspective. Firstly, under swarm intelligence principles, bioinspired search engines benefit from advanced mechanisms that consider the knowledge gathered by all the individuals in the swarm to improve solution quality. Secondly, the introduction of quality indicators for fitness assignment purposes represents a significant tool to attain better search guidance in this problem. As a result, the combination of both strategies provides a suitable approach to acquire advanced search capabilities, giving rise to an algorithmic design, IMOBA, which was able to achieve the most satisfying multiobjective performance.

Regarding the biological assessment of the obtained results, our experimentation showed the benefits associated to the formulation of phylogenetic inference as a MOP, whose implications can be discussed from different perspectives. The first challenge to be addressed is on the accuracy of multiobjective optimizers from a single-objective perspective, that is, the assessment of the best parsimony and best likelihood trees in the Pareto front with regard to other phylogenetic tools. By making comparisons under a biostatistical framework, we found the our multiobjective approaches (represented by IMOBA as the best indicator-based design and MOABC as the best dominance-based design) succeeded in generating high-quality phylogenies attending to each considered objective function separately. More specifically, the evaluation of parsimony quality suggested that our approaches were able to match the parsimony scores of the phylogenies provided by the state-of-the-art method

TNT, while improving the scores reported by the traditional tool DNAPARS. From a likelihood perspective, comparisons with widely-used methods, such as RAxML, Garli, IQPNNI, and MrBayes, gave account of how our approaches achieved the best likelihood scores in most of the data sets under study. Regarding other multiobjective methods from the literature, comparisons with PhyloMOEA also pointed out the quality of the phylogenies inferred by our proposals.

The second major challenge that a multiobjective approach must face is the justification of the biological relevance of the multiobjective phylogenies contained in the Pareto front. The biological usefulness of such multiobjective hypotheses, which are obtained as a result of the simultaneous optimization of the studied objective functions, can be assessed from two perspectives. Firstly, those phylogenies inferred under multiobjective principles represent a compromise answer to the conflicts which arise when parsimony and likelihood disagree about the best supported evolutionary histories. Particularly, by considering the simultaneous optimization of parsimony and likelihood, these topologies are able to include evolutionary relationships with biological sense attending to both criteria which are not satisfactorily recovered by single-criterion trees separately. This idea was supported by our experimentation on a well-known Plethodontid salamander dataset, in which the analyzed multiobjective topology successfully dealt with the conflicts reported by single-criterion analyses, while also identifying major salamander families in the phylogeny according to the taxonomy of these organisms. Secondly, the information on evolutionary relationships provided by all the inferred Pareto solutions can be combined into a single consensus phylogeny, which is useful to provide more reliable conclusions from a statistical and biological point of view. This idea was also applied in this Thesis, reporting consensus multiobjective trees generated from the fronts inferred by IMOBA.

Therefore, this research has given account of the promising results which suppose the application of multiobjective optimization to phylogenetics. By using this kind of techniques, phylogenetic search engines are able to generate high-quality solutions attending to multiple criteria in a single run (hence simplifying experimentation for biologists), while also providing a set of Pareto trees to address incongruence issues and introduce more robustness into phylogenetic analyses.

Conclusions on the Application of Parallelism to Phylogenetics

Due to the computational complexity of the problem, a key goal in this Thesis laid on the application of parallel computing to undertake time-consuming phylogenetic inferences. In this context, the proposed line of work was focused on the application of intra-algorithm parallelization to our multiobjective metaheuristics on high performance computing environments, along with exploring the use of heterogeneous computing to study problem dependent fine-grained parallelization techniques. In order to parallelize our metaheuristics, we studied parallel designs for shared-memory systems and hybrid shared-distributed memory architectures. OpenMP-based parallel implementations of our metaheuristics were described, bearing in mind two main parallelization schemes for pure shared-memory environments: synchronous generational designs and asynchronous non-generational designs. In addition, mixed mode MPI+OpenMP implementations based on the master-worker model

were proposed to take advantage of the parallel capabilities of multicore clusters and constellation-like systems. As for heterogeneous computing, we addressed the parallelization of the phylogenetic parsimony function on CPU+GPU systems with OpenCL, implementing optimized GPU and CPU kernels and proposing a multidevice design to fully exploit the underlying heterogeneous resources.

Our experimentation on shared-memory models followed two main directions. In the first place, a comparative study on synchronous designs for the six metaheuristics applied in this Thesis was conducted, assessing parallel performance by using the speedup and efficiency metrics. The comparison showed the meaningful parallel results obtained by MO-FA, representing the metaheuristic which attained the most satisfying exploitation of shared-memory resources under synchronous parallelization schemes (efficiencies in the range 73.23% - 89.95% for 24 cores). IBEA, NSGA-II, and SPEA2 also presented good parallel results, suggesting the suitability of these evolutionary designs to be improved by means of parallel computing. On the other hand, the results reported by IMOBA and MOABC revealed that the parallelization of these metaheuristics represented a more challenging question due to their algorithmic features. Using MOABC as case study, we provided an alternative parallelization scheme inspired by the asynchronous parallel behaviour of honey bees. This nature-inspired approach succeed in solving the main performance pitfalls shown by the synchronous design, obtaining an almost optimal performance on a shared-memory system composed of 48 cores (leading to efficiencies up to 93.25%). In addition, comparisons of hypervolume performance between the synchronous and asynchronous implementations of MOABC highlighted that the parallel asynchronous approach did not lead to a worsening in the search capabilities of the algorithm. Finally, a comparative evaluation of parallel scalability with other parallel phylogenetic methods (RAxML, IQPNNI, MrBayes, and PhyloMOEA) also supported the relevance of the asynchronous design. As a result, we can claim that this kind of asynchronous non-generational models represents a promising strategy to parallelize metaheuristics, leading to an accurate exploitation of shared-memory environments.

With regard to the assessment of mixed mode approaches, two studies were undertaken. In the first one, we analyzed the parallel results achieved on a multicore cluster system by two synchronous algorithmic designs with different characteristics: MO-FA (representing a swarm intelligence algorithm with dynamic workload per generation) and NSGA-II (representing a MOEA with static workload). For this purpose, we examined the different factors that impact the performance of hybrid parallel systems, such as the parallel efficiency achieved at worker computations, overhead times due to synchronizations and communications, and other critical and master times. This study showed that MO-FA was able to fit the characteristics of the considered cluster architecture, improving the efficiency observed at the worker task processing level and benefiting from the introduction of dynamic loop scheduling policies. As a consequence, MO-FA achieved efficiencies up to 71.28% on configurations involving 48 cores, improving in a statistically significant way the efficiencies reported by NSGA-II (66.76% in its best case). The second study conducted the evaluation of the mixed mode implementation of IBEA on a constellation system. An accurate exploitation of such systems requires an optimal configuration of the hybrid approach attending to the number of MPI processes

and OpenMP threads per process. By examining different configurations of MPI processes/OpenMP threads, we found that IBEA reported the best speedup values when one MPI process was allocated at each processor, relying in OpenMP to exploit their multicore capabilities. In this way, the proposed implementation was able to benefit from the strengths of both parallel programming models, reporting efficiencies up to 89.51% at the intra-node level (24 cores) and 78.81% when using all the clustered resources (48 cores). As in the shared-memory case, the relevance of these results was ensured by performing satisfactory comparisons with other hybrid parallel phylogenetic methods.

The last experimental scenario in this Thesis addressed the evaluation of fine-grained parallel techniques to accelerate objective function computations on heterogeneous CPU+GPU systems. Considering the parallelization of parsimony evaluations under Fitch's algorithm, we verified the relevant role that current heterogeneous platforms can play in the process of accelerating time-consuming analyses on very complex data sets. For experimentation purposes, six real data sets with different features (in terms of number of species and sequence lengths) were considered, containing sequences from enterobacteriaceae, salmonella, amphibian and mammalian organisms, green plants, and HIV1. Our experimental results pointed out the relationship between the characteristics of the input data and the performance achieved by the examined computing devices when making kernel calculations. While the processing of complex data sets involving large sequence length benefited from the high data parallel capabilities of GPU-based accelerators, multicore CPUs can also make relevant contributions to accelerate this objective function and even outperform GPUs on alignments with limited sequence length. This is the reason why a heterogenous CPU+GPU multidevice design represents a suitable approach to deal with computationally demanding evaluation procedures in phylogenetics. By taking full advantage of the capabilities of CPU+multiGPU platforms, our proposal achieved very significant parallel results over the serial version of Fitch's algorithm, showing speedups up to 615x/800x. Comparisons with the fastest open-source implementation of the parsimony function, Parsimonator, confirmed the success of applying heterogeneous computing to accelerate evaluation steps in complex optimization problems like phylogenetic inference.

11.2 Future Work Lines

The findings obtained in this PhD Thesis lay the foundations of a wide variety of future research directions. Some suggestions on future work lines are detailed below:

1. From a multiobjective and biological perspective, different lines can be distinguished. Firstly, we are interested in evaluating an increasingly popular trend in multiobjective algorithmic development based on the decomposition of a MOP into different multiobjective optimization subproblems [192]. This kind of decomposition-based multiobjective optimizers will be compared with the ones applied in this Thesis, along with integrating swarm intelligence mechanisms to improve solution quality. Another way to improve the biological accuracy of the inferred phylogenies lies on applying non-homogeneous models of nucleotide substitution [55],

introducing more customization to the inference process. Parallelism can also be integrated to improve multiobjective performance, exploring self-contained parallel cooperation techniques to implement island-based models and parallel teams of bioinspired algorithms. Finally, an additional line of work is focused on the assessment of our best bioinspired metaheuristic, IMOBA, on three-dimensional and many-objective optimization problems.

2. From a parallel computing perspective, an immediate line of work consists of applying asynchronous parallelization models to IMOBA, in order to integrate the most efficient parallelization scheme examined in this Thesis into the metaheuristic which achieved the best multiobjective and biological results. The idea is to study the natural behaviour of bats to find some patterns that could be applied to model the asynchronous non-generational scheme, extending the experimentation to shared-memory systems composed of 64 cores. In addition, the porting of the resulting design to supercomputer platforms represents a promising approach to address large-scale multiobjective phylogenomic analyses. Asynchronous parallel metaheuristics will also be evaluated on multicore clusters, in order to find out if the conclusions obtained on shared memory are translatable to this kind of parallel technologies. Regarding the application of heterogeneous computing, we aim to undertake comparisons between the standard OpenCL and vendor-specific programming libraries [187] to maximize the exploitation of CPU+GPU systems. The proposed kernels will be extended to support more complex molecular data such as protein sequences, which introduce additional complexity due to the need of applying computations considering 20 different amino acids per sequence site.

3. Regarding the application of bioinspired computing and parallelism to other problems, several directions can be defined. In the phylogenetics field, promising advances could be achieved by introducing multiobjective optimization techniques to solve uncertainty in model selection. The idea is to conduct the search for multiobjective phylogenies that represent a tradeoff between model-based criteria assuming different models of sequence evolution. In this way, the relationship between the characteristics of the input alignment and the presence of model uncertainty can be studied by examining the different topologies contained in the Pareto front. Other possible applications of bioinspired computing and parallelism to phylogenetics are related to the inference of supertrees [60] and phylogenetic networks [86]. Finally, other lines involve the application of the techniques studied in this Thesis to other bioinformatics applications, such as protein structure prediction [134] and gene regulatory network inference [79].

11.3 Summary

This chapter has summarized the main findings of this PhD Thesis. From a multiobjective performance perspective, this research has given account of the benefits of combining swarm intelligence and indicator-based fitness assignment procedures to improve search capabilities. From a biological point of view, the formulation of phylogenetic inference as a MOP has provided a number of use-

ful applications to introduce more robustness to the inference process. Finally, the study of parallel computing models has shown the relevance of undertaking asynchronous non-generational parallelization schemes on shared-memory systems, mixed mode implementations which accurately fit the characteristics of multicore clusters, and heterogeneous computing to take advantage of the data parallelism opportunities shown by biological objective functions. From these conclusions, a wide number of future work lines have been defined, ranging from further developments on phylogenetic inference to the application of the methods proposed in this Thesis to other optimization problems.

Chapter 12

Scientific Achievements of the Research

This chapter summarizes the scientific achievements obtained throughout the development of this PhD Thesis. As a result of the relevant contributions associated to the research, a total of 24 publications have been attained, including 5 publications in international journals indexed in the Journal Citation Reports (JCR), 5 book chapters, 6 publications in IEEE/ACM/Springer international conferences, and 8 national conferences. In addition, we report additional scientific achievements related to the participation of the PhD candidate in research projects, research grants, special issues, international stays, reviewer tasks in international journals, organization of conferences, workshops, and special sessions, collaborations with other institutions, teaching, and other mentions.

12.1 Scientific Publications

One of the main goals of this PhD Thesis was to guarantee the quality of the research by carrying out the discussion and diffusion of the obtained results to the scientific community. For this purpose, we have considered the usual mechanisms of scientific publication through international peer-reviewed JCR-indexed journals and international/national peer-reviewed conferences related to the topics of this research. As a result, a total of 24 publications have been obtained, ensuring the relevance of the research undertaken in this Thesis. These publications are listed below.

A total of 5 publications in international JCR-indexed journals have been achieved:

1. On the Design of Shared Memory Approaches to Parallelize a Multiobjective Bee-Inspired Proposal for Phylogenetic Reconstruction. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez. Information Sciences, Volume 324, Elsevier Science, New York, NY, USA, 2015, pp. 163-185, ISSN: 0020-0255. (Impact factor = 4.038 in 2014, Quartile Q1)

2. Parallel Multiobjective Metaheuristics for Inferring Phylogenies on Multicore Clusters. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez. IEEE Transactions on Parallel and Distributed Systems, Volume 26, Issue 6, IEEE Computer Society, Los Alamitos, CA, USA, 2015, pp. 1678-1692, ISSN: 1045-9219. (Impact factor = 2.170 in 2014, Quartile Q1)

3. A Hybrid Approach to Parallelize a Fast Non-Dominated Sorting Genetic Algorithm for Phylogenetic Inference. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez. Concurrency and Computation: Practice and Experience, Volume 27, Issue 3, Wiley-Blackwell, Malden, England, 2015, pp. 702-734, ISSN: 1532-0626. (Impact factor = 0.997 in 2014, Quartile Q2)

4. Applying a Multiobjective Metaheuristic Inspired by Honey Bees to Phylogenetic Inference. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez. BioSystems, Volume 114, Issue 1, Elsevier Science, Oxford, England, 2013, pp. 39-55, ISSN: 0303-2647. (Impact factor = 1.472, Quartile Q2)

5. Parallelism-based Technologies in Bioinformatics and Biomedicine: A View from Diverse Perspectives. Miguel A. Vega-Rodríguez, **Sergio Santander-Jiménez**. Concurrency and Computation: Practice and Experience, Wiley-Blackwell, Malden, England, 2015, pp. 1-3, ISSN: 1532-0626. (Impact factor = 0.997 in 2014, Quartile Q2) (*Accepted*, doi: http://dx.doi.org/10.1002/cpe.3596)

A total of 5 book chapters in the Lecture Notes in Computer Science (LNCS) series have been published:

1. Inferring Multiobjective Phylogenetic Hypotheses by Using a Parallel Indicator-Based Evolutionary Algorithm, in: Theory and Practice of Natural Computing, LNCS, Vol. 8890. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez. Springer, Switzerland, 2014, pp. 205-217. ISBN: 978-3-319-13748-3.

2. A Multiobjective Proposal Based on the Firefly Algorithm for Inferring Phylogenies, in: Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, LNCS, Vol. 7833. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez. Springer-Verlag, Berlin Heidelberg, Germany, 2013, pp. 141-152. ISBN: 978-3-642-37188-2.

3. A Parallel Multiobjective Algorithm Inspired by Fireflies for Inferring Evolutionary Trees on Multicore Machines, in: Computer Aided Systems Theory, LNCS, Vol. 8111. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez. Springer-Verlag, Berlin Heidelberg, Germany, 2013, pp. 412-419. ISBN: 978-3-642-53855-1.

4. Comparing Different Operators and Models to Improve a Multiobjective Artificial Bee Colony Algorithm for Inferring Phylogenies, in: Theory and Practice of Natural Computing, LNCS, Vol. 7505. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez, Juan A. Gómez-Pulido, Juan M. Sánchez-Pérez. Springer-Verlag, Berlin Heidelberg, Germany, 2012, pp. 187-200. ISBN: 978-3-642-33859-5.

5. Inferring Phylogenetic Trees Using a Multiobjective Artificial Bee Colony Algorithm, in: Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, LNCS, Vol. 7246. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez, Juan A. Gómez-Pulido, Juan

M. Sánchez-Pérez. Springer-Verlag, Berlin Heidelberg, Germany, 2012, pp. 144-155. ISBN: 978-3-642-29065-7.

A total of 14 conference papers, including 6 IEEE/ACM/Springer international conferences and 8 well-known Spanish national conferences, have been published:

1. Applying OpenMP-based Parallel Implementations of NSGA-II and SPEA2 to Study Phylogenetic Relationships. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez. Proceedings of the 2014 IEEE International Conference on Cluster Computing (IEEE Cluster 2014), IEEE Computer Society, Madrid, Spain, 2014, pp. 305-313. ISBN: 978-1-4799-5547-3.

2. Performance Analysis of Multiobjective Artificial Bee Colony Implementations for Phylogenetic Reconstruction. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez. Proceedings of the Sixth World Congress on Nature and Biologically Inspired Computing, IEEE Computer Society, Porto, Portugal, 2014, pp. 35-40. ISBN: 978-1-4799-5937-2.

3. A Comparative Study on Distance Methods Applied to a Multiobjective Firefly Algorithm for Phylogenetic Inference. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez. Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO 2013), Association for Computing Machinery (ACM), Amsterdam, The Netherlands, 2013, pp. 1587-1594. ISBN: 978-1-4503-1964-5.

4. Parallelizing a Multiobjective Swarm Intelligence Approach to Phylogenetics Using Hybrid MPI/OpenMP Schemes. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez. Proceedings of the 20th EuroMPI 2013, Association for Computing Machinery (ACM), Madrid, Spain, 2013, pp. 193-198. ISBN: 978-84-616-5133-7.

5. Using OpenMP to Parallelize a Multiobjective Firefly Algorithm for Phylogenetic Inference. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez, Juan A. Gómez-Pulido, Juan M. Sánchez-Pérez. Computer Aided Systems Theory - Extended Abstracts, A. Quesada, J.C. Rodriguez, R. Moreno jr., R. Moreno (Eds.). IUCTC. Universidad de Las Palmas de Gran Canaria, Las Palmas de Gran Canaria, Spain, 2013, pp. 239-241. ISBN: 978-84-695-6971-9.

6. Evaluating the Performance of a Parallel Multiobjective Artificial Bee Colony Algorithm for Inferring Phylogenies on Multicore Architectures. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez, Juan A. Gómez-Pulido, Juan M. Sánchez-Pérez. Proceedings of the 10th IEEE International Symposium on Parallel and Distributed Processing with Applications (IEEE ISPA 2012), IEEE Computer Society, Madrid, Spain, 2012, pp. 713-720. ISBN: 978-0-7695-4701-5.

7. Análisis Comparativo de Implementaciones del Algoritmo Multiobjective Artificial Bee Colony para Reconstrucción Filogenética. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez.

Actas del X Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, Universidad de Extremadura, Mérida, Spain, 2015, pp. 369-376. ISBN: 978-84-697-2150-6.

8. Usando GPUs para Acelerar Reconstrucciones Filogenéticas bajo el Criterio de Parsimonia. **Sergio Santander-Jiménez**, Aleksandar Ilic, Leonel Sousa, Miguel A. Vega-Rodríguez. Actas de las XXVI Jornadas de Paralelismo (JP 2015), Universidad de Córdoba, Córdoba, Spain, 2015, pp. 38-44. ISBN: 978-84-16017-52-2.

9. Estudiando Relaciones Filogenéticas Mediante Metaheurísticas Multiobjetivo Paralelas Basadas en OpenMP. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez. Actas de las XXV Jornadas de Paralelismo (JP 2014), Universidad de Valladolid, Valladolid, Spain, 2014, pp. 173-178. ISBN: 978-84-697-0329-3.

10. Paralelizando una Aproximación Multiobjetivo y Bioinspirada para Filogenética Usando Esquemas Híbridos MPI/OpenMP. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez. Actas de las XXIV Jornadas de Paralelismo (JP 2013), Servicio de Publicaciones, Universidad Complutense de Madrid, Madrid, Spain, 2013, pp. 253-258. ISBN: 978-84-695-8330-2.

11. Una Aproximación Multiobjetivo Basada en el Comportamiento de las Luciérnagas para la Inferencia Filogenética. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez. Actas del IX Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, J. M. Colmenar et al. (editores), Universidad Complutense de Madrid y Universidad Rey Juan Carlos, Madrid, Spain, 2013, pp. 614-623. ISBN: 978-84-695-8348-7.

12. Evaluando el Rendimiento Multicore de una Aproximación Multiobjetivo de Inteligencia de Enjambre para la Inferencia Filogenética. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez, Juan A. Gómez-Pulido, Juan M. Sánchez-Pérez. Actas de las XXIII Jornadas de Paralelismo (JP 2012), Servicio de Publicaciones, Universidad Miguel Hernández, Elche, Spain, 2012, pp. 98-103. ISBN: 978-84-695-4471-6.

13. Un Sistema Híbrido MPI/OpenMP basado en el Algoritmo NSGA-II para la Inferencia de Árboles Filogenéticos. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez, Juan A. Gómez-Pulido, Juan M. Sánchez-Pérez. Actas de las XXIII Jornadas de Paralelismo (JP 2012), Servicio de Publicaciones, Universidad Miguel Hernández, Elche, Spain, 2012, pp. 122-127. ISBN: 978-84-695-4471-6.

14. Una Adaptación Multiobjetivo y Paralela del Algoritmo Artificial Bee Colony Aplicada a la Inferencia Filogenética. **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez, Juan A. Gómez-Pulido, Juan M. Sánchez-Pérez. Actas del VIII Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, J. A. Gámez, J. M. Puerta, F. Parreño y L. de la Ossa (editores), Universidad de Castilla-La Mancha, Albacete, Spain, 2012, pp. 375-382. ISBN: 978-84-615-6931-1.

12.2 Other Related Scientific Achievements

This section enumerates other scientific achievements obtained by the candidate throughout the development of this PhD Thesis. These merits, which include his participation in research projects, research grants, editor of special issues, stays in international research centres, reviewer of international JCR-indexed journals, organizer of conferences, workshops, and special sessions, collaborations with international institutions, teaching, and mentions, are listed and detailed below:

12.2.1 Participation in Research Projects

Sergio Santander-Jiménez has participated as a researcher in the following research projects:

- BIO TIN2012-30685 (2013 - present)
  - Project title: BIO: Optimización Multiobjetivo y Paralelismo en Bioinformática.
  - Funding institution: Spanish Ministry of Economy and Competitiveness (Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica).
  - Participating institutions: University of Extremadura.
  - Duration: 3 years (2013 - 2015).
  - Budget: 107,160 euros.
  - Main researcher: Miguel A. Vega-Rodríguez.
  - Number of researchers: 12.
  - Summary: This project is aimed at innovating in multiple fronts of multiobjective optimization, parallelism and bioinformatics. For this purpose, we plan to advance in fundamental research by developing new multiobjective models for algorithms and other procedures capable of solving problems of realistic dimension and complexity, improving, creating and disseminating advanced multiobjective algorithms. Furthermore, we will apply parallelism-based technologies (in their different varieties: multi-core computing, cluster computing, grid computing, hardware accelerators...) to the multiobjective resolution of complex problems, improving their results, performance and throughput. The scalability and efficiency of the parallel techniques developed will be also studied, including comparative studies, and analyzing the possible combination of different parallel technologies and parallelism levels to solve a same problem. The problems tackled are not the typical instances drawn from benchmarks, but instead we will address real-world problems from the hot domain of bioinformatics (applied research). This way, the benefits will lie in both methodology and real applications. The goal is to improve the efficiency and effectivity of the solutions to bioinformatics problems with respect to the present state of the art; we aim at showing that the contributed techniques are not only appealing in theory, but also effective and useful for society. The bioinformatics

problems to address are: motif discovery, phylogenetic inference, and design of DNA sequences for DNA computing. These are important and real problems in the bioinformatics domain, representing by themselves interesting project outcomes. We will tackle a clear mission for internationalization of the results, with high impact publications, research staff training, visits to foreign teams and of foreign researchers to our headquarter, organization of talks, workshops, and special issues in international journals.

12.2.2 Research Grants

Sergio Santander-Jiménez has benefited from the following research grants, which were obtained on competitive basis:

1. Programa Nacional de Formación de Profesorado Universitario, FPU (FPU12/04101). Spanish Ministry of Education, Culture, and Sports. From March 2013 to November 2015.

2. Postgraduate grant from the Fundación Fernando Valhondo Calaff. From January 2012 to February 2013.

3. Postgraduate grant from the University of Extremadura (Plan de Iniciación a la Investigación, Desarrollo Tecnológico e Innovación, Acción II). From November 2011 to December 2011.

12.2.3 Editor of Books and Journals

Sergio Santander-Jiménez has served as editor of the following special issue for the journal Concurrency and Computation: Practice and Experience:

- Special Issue on Parallelism-based Technologies in Bioinformatics and Biomedicine (Concurrency and Computation: Practice and Experience). Miguel A. Vega-Rodríguez, **Sergio Santander-Jiménez**. Wiley-Blackwell, Malden, England, 2015. ISSN: 1532-0626. (Impact factor = 0.997 in 2014, Quartile Q2)

12.2.4 Stays in International Research Centres

Sergio Santander-Jiménez visited the R&D unit of the Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento em Lisboa (INESC-ID) in Lisbon, Portugal, for a period of 3 months (from September 1, 2014 to December 1, 2014). During his research stay, he worked on the application of heterogeneous GPU+CPU computing to address biological optimization problems.

12.2.5 Reviewer of International JCR-Indexed Journals

Sergio Santander-Jiménez is currently serving as a reviewer for the following international JCR-indexed journals:

1. Applied Soft Computing, Elsevier. ISSN 1568-4946. Impact factor = 2.810 (in 2014). Date: from February 2014 to present.

2. Parallel Computing, Elsevier. ISSN 0167-8191. Impact factor = 1.511 (in 2014). Date: from December 2013 to present.

3. Soft Computing, Springer. ISSN 1432-7643. Impact factor = 1.271 (in 2014). Date: from April 2014 to present.

4. Concurrency and Computation: Practice and Experience, Wiley. ISSN 1532-0626. Impact factor = 0.997 (in 2014). Date: from December 2014 to present.

5. Journal of Universal Computer Science, Springer. ISSN 0948-695X. Impact factor = 0.466 (in 2014). Date: from October 2012 to present.

6. Journal of Systems Architecture, Elsevier. ISSN 1383-7621. Impact factor = 0.440 (in 2014). Date: from December 2012 to present.

12.2.6 Organizer of Conferences, Workshops, and Special Sessions

Sergio Santander-Jiménez has taken active role as organizing committee, program committee, local committee, and session chair for several conferences, workshops, and special sessions:

1. Third International Workshop on Parallelism in Bioinformatics (PBio2015, IEEE ISPA 2015 workshop). August 2015, Helsinki, Finland. Role: organizing committee, program committee.

2. Second Congress on Multicore and GPU Programming (PPMG2015). March 2015, Cáceres, Spain. Role: local committee.

3. Special session on Metaheuristics, Evolutionary and Bioinspired Algorithms in Bioinformatics (MAEB 2015). February 2015, Mérida, Spain. Role: organizing committee, program committee, session chair.

4. Second International Workshop on Parallelism in Bioinformatics (PBio2014, IEEE Cluster 2014 workshop). September 2014, Madrid, Spain. Role: organizing committee, program committee, session chair.

5. 2nd International Conference on the Theory and Practice of Natural Computing (TPNC 2013, LNCS). December 2013, Cáceres, Spain. Role: local committee.

6. First International Workshop on Parallelism in Bioinformatics (PBio2013, ACM EuroMPI 2013 workshop). September 2013, Madrid, Spain. Role: organizing committee, program committee, session chair.

7. Special session on Metaheuristics, Evolutionary and Bioinspired Algorithms in Bioinformatics (MAEB 2013). September 2013, Madrid, Spain. Role: organizing committee, program committee, session chair.

8. International Workshop on Evolutionary Computation in Bioinformatics (BIO2013, ACM GECCO 2013 workshop). July 2013, Amsterdam, The Netherlands. Role: organizing committee, program committee.

12.2.7 Collaborations with International Institutions

Sergio Santander-Jiménez has worked in collaboration with researchers from the Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento em Lisboa (INESC-ID) during his research stay, studying the application of heterogeneous CPU+GPU computing to parallelize time-consuming optimization procedures in bioinformatics.

12.2.8 Teaching

In the last two years, Sergio Santander-Jiménez has actively participated in teaching tasks for the Department of Computer and Communications Technologies, University of Extremadura. More specifically, he has collaborated through *Venia Docendi* (from his FPU grant) in the teaching of the following subjects:

1. High Performance Computing (Computación de Altas Prestaciones). Master's degree in Computer Science. Years: 2013/2014 and 2014/2015.

2. Computer Architecture (Arquitectura de Computadores). Degree in Computer Engineering. Years: 2013/2014 and 2014/2015.

12.2.9 Mentions and Other Achievements

Other achievements and mentions obtained by Sergio Santander-Jiménez throughout the development of this Thesis are the following:

1. Spanish Parallel Programming Contest. Record in the 2014 'F' problem: Optimal assignation of resources to attention centers with OpenMP (parallel optimization problem), http://luna.inf.um.es/2015/records.php.

2. Best paper award nominee: Inferring Phylogenetic Trees Using a Multiobjective Artificial Bee Colony Algorithm, by **Sergio Santander-Jiménez**, Miguel A. Vega-Rodríguez, Juan A. Gómez-Pulido, and Juan M. Sánchez-Pérez. EVO* 2012 (EVOBIO), the main European events on Evolutionary Computation. 11-13 April 2012, Málaga, Spain.

12.3 Summary

This chapter has reported the main scientific achievements of the research carried out in this PhD Thesis. A total of 24 publications give support to the significant contributions which represent the findings of this thesis, including 5 JCR-indexed journal papers, 5 book chapters, and 14 communications to conferences with international and national scope. In addition, this chapter has summarized other achievements related to the participation of the PhD candidate in research projects, stays and collaborations with international institutions, contributions as guest editor and reviewer in international JCR-indexed journals, conference organizing committees, teaching, and mentions.

References

[1] (2005). HIV Sequence Database. http://www.hiv.lanl.gov/.

[2] (2015). TOP500 supercomputer sites. http://top500.org/.

[3] Acosta, A., Blanco, V., and Almeida, F. (2012). Towards the Dynamic Load Balancing on Heterogeneous Multi-GPU Systems. In *Proc. of the 10th IEEE International Symposium on Parallel and Distributed Processing with Applications*, pages 646–653. IEEE Computer Society.

[4] Adhianto, L. and Chapman, B. (2007). Performance modeling of communication and computation in hybrid MPI and OpenMP applications. *Simulation Modelling Practice and Theory*, 15(4):481–491.

[5] Alachiotis, N. and Stamatakis, A. (2011). FPGA Acceleration of the Phylogenetic Parsimony Kernel? In *Proc. of the 21st International Conference on Field Programmable Logic and Applications*, pages 417–422. IEEE Computer Society.

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

[7] Ayres, D. L. et al. (2012). BEAGLE: An Application Programming Interface and High-Performance Computing Library for Statistical Phylogenetics. *Systematic Biology*, 61(1):170–173.

[8] Bader, D. A., Chandu, V. P., and Yan, M. (2006a). ExactMP: An Efficient Parallel Exact Solver for Phylogenetic Tree Reconstruction Using Maximum Parsimony. In *Proc. of the 2006 International Conference on Parallel Processing*, pages 65–73. IEEE Computer Society.

[9] Bader, D. A., Stamatakis, A., and Tseng, C. W. (2006b). Computational Grand Challenges in Assembling the Tree of Life: Problems and Solutions. In *Advances in Computers*, volume 68, pages 127–176. Elsevier.

[10] Bai, A. (2014). Multiobjective Clustering Using Support Vector Machine: Application to Microarray Cancer Data. In *Intelligent Computing, Networking, and Informatics*, volume 243 of *Advances in Intelligent Systems and Computing*, pages 1209–1215. Springer India.

[11] Bao, J., Xia, H., Zhou, J., Liu, X., and Wang, G. (2013). Efficient Implementation of MrBayes on Multi-GPU. *Molecular Biology and Evolution*, 30(6):1471–1479.

[12] Barker, D. (2003). LVB: parsimony and simulated annealing in the search for phylogenetic trees. *Bioinformatics*, 20(2):274–275.

[13] Beni, G. (1988). The Concept of Cellular Robotic Systems. In *Proc. of the IEEE International Symposium on Intelligent Systems*, pages 57–62. IEEE Computer Society.

[14] Beume, N., Fonseca, C. M., López-Ibáñez, M., Paquete, L., and Vahrenhold, J. (2009). On the Complexity of Computing the Hypervolume Indicator. *IEEE Transactions on Evolutionary Computation*, 13(5):1075–1082.

[15] Blum, C. and Merkle, D. (2008). *Swarm Intelligence - Introduction and Applications*. Springer-Verlag, Berlin Heidelberg.

[16] Bos, D. H. and Posada, D. (2005). Using models of nucleotide evolution to build phylogenetic trees. *Developmental and Comparative Immunology*, 29(3):211–227.

[17] Bova, S., Breshears, C., Eigenmann, R., Gabb, H., Gaertner, G., Kuhn, B., Magro, B., Salvini, S., and Vatsa, V. (1999). Combining Message-Passing and Directives in Parallel Applications. *SIAM News*, 32(9):10–14.

[18] Brauer, M. J., Holder, M. T., Dries, L. A., Zwickl, D. J., Lewis, P. O., and Hillis, D. M. (2002). Genetic algorithms and parallel processing in maximum-likelihood phylogeny inference. *Molecular Biology and Evolution*, 19(10):1717–1726.

[19] Bryant, D. (2003). A Classification of Consensus Methods for Phylogenies. In *BioConsensus*, DIMACS series in discrete mathematics and theoretical computer science, pages 163–184. American Mathematical Society.

[20] Bryant, D., Galtier, N., and Poursat, M. A. (2005). Likelihood calculations in molecular phylogenetics. In *Mathematics of Evolution and Phylogeny*, pages 33–62. Oxford University Press.

[21] Burjorjee, K. M. (2013). Explaining optimization in genetic algorithms with uniform crossover. In *Proc. of the Twelfth Workshop on Foundations of Genetic Algorithms*, pages 37–50. Association for Computing Machinery (ACM).

[22] Burleigh, J. G. and Mathews, S. (2007). Assessing Systematic Error in the Inference of Seed Plant Phylogeny. *International Journal of Plant Sciences*, 168(2):125–135.

[23] Cancino, W. and Delbem, A. C. B. (2007). A Multi-Objective Evolutionary Approach for Phylogenetic Inference. In *Evolutionary Multi-Criterion Optimization*, volume 4403 of *LNCS*, pages 428–442. Springer-Verlag.

[24] Cancino, W. and Delbem, A. C. B. (2010). A Multi-Criterion Evolutionary Approach Applied to Phylogenetic Reconstruction. In *New Achievements in Evol. Comp.*, pages 135–156. InTech.

[25] Cancino, W., Jourdan, L., Talbi, E. G., and Delbem, A. C. B. (2010). Parallel Multi-Objective Approaches for Inferring Phylogenies. In *Evolutionary Computation, Machine Learning and Data Mining in Computational Biology*, volume 6023 of *LNCS*, pages 26–37. Springer-Verlag.

[26] Ceron, C., Dopazo, J., Zapata, E. L., Carazo, J. M., and Trelles, O. (1998). Parallel Implementation for DNAml Program on Message-Passing Architectures. *Parallel Computing*, 24(5-6):701–716.

[27] Chai, J., Su, H., Wen, M., Cai, X., Wu, N., and Zhang, C. (2013). Resource-efficient utilization of CPU/GPU-based heterogeneous supercomputers for Bayesian phylogenetic inference. *The Journal of Supercomputing*, 66(1):364–380.

[28] Chai, J., Su, H., and Zhang, C. (2012). Performance Analysis and Comparison of Three MrBayes Computational Biology Code on TianHe-1A Supercomputer. In *Proc. of the International Conference on Computer Science and Service System 2012*, pages 2135–2140. IEEE Computer Society.

[29] Chapman, B., Jost, G., and van der Pas, R. (2007). *Using OpenMP: Portable Shared Memory Parallel Programming*. The MIT Press, Cambridge, Massachusetts.

[30] Chase, M. W. et al. (1993). Phylogenetics of seed plants: An analysis of nucleotide sequences from the plastid gene rbcl. *Annals of the Missouri Botanical Garden*, 80(3):528–580.

[31] Chassagnole, C., Rodriguez, J. A., Doncescu, A., and Yang, L. T. (2006). Differential Evolutionary Algorithms for In Vivo Dynamic Analysis of Glycolysis and Pentose Phosphate Pathway in Escherichia coli. In *Parallel Computing for Bioinformatics and Computational Biology: Models, Enabling Technologies, and Case Studies*, pages 59–78. John Wiley & Sons Inc.

[32] Chippindale, P. T., Bonett, R. M., Baldwin, A. S., and Wiens, J. J. (2004). Phylogenetic evidence for a major reversal of life history evolution in plethodontid salamanders. *Evolution*, 58(12):2809–2822.

[33] Chor, B. and Tuller, T. (2005). Maximum likelihood of evolutionary trees: hardness and approximation. *Bioinformatics*, 21(1):97–106.

[34] Chu, D. and Zomaya, A. Y. (2006). Parallel Ant Colony Optimization for 3D Protein Structure Prediction Using the HP Lattice Model. In *Parallel Evolutionary Computations*, volume 22 of *Studies in Computational Intelligence*, pages 177–198. Springer Berlin.

[35] Coelho, G. P., Silva, A. E. A., and Zuben, F. J. V. (2010). An Immune-Inspired Multi-Objective Approach to the Reconstruction of Phylogenetic Trees. *Neural Computing and Applications*, 19(8):1103–1132.

[36] Coello, C., Dhaenens, C., and Jourdan, L. (2010). *Advances in Multi-Objective Nature Inspired Computing*. Springer-Verlag, Berlin Heidelberg.

[37] Coello, C., Veldhuizen, D. V., and Lamont, G. (2002). Evolutionary algorithms for solving multi-objective problems. In *Genetic Algorithms and Evolutionary Computation*, volume 5. Kluwer Academic Publishers.

[38] Cole, J. R. et al. (2005). The Ribosomal Database Project (RDP-II): sequences and tools for high-throughput rRNA analysis. *Nucleic Acids Research*, 33 (Database issue D294-D296).

[39] Congdon, C. B. (2002). Gaphyl: An Evolutionary Algorithms Approach for the Study of Natural Evolution. In *Genetic and Evolutionary Computation Conference*, pages 1057–1064.

[40] Congdon, C. B. (2008). Phylogenetic Inference Using Evolutionary Algorithms. In *Computational Intelligence in Bioinformatics*, pages 237–262. Wiley-IEEE Press.

[41] Congdon, C. B. and Greenfest, E. F. (2000). Gaphyl: A genetic algorithm approach to cladistics. In *Data Mining with Evolutionary Algorithms (A. A. Freitas, W. Hart, N. Krasnogor, J. Smith, eds.)*, pages 85–88.

[42] Congdon, C. B. and Septor, K. J. (2003). Phylogenetic Trees using Evolutionary Search: Initial Progress in Extending Gaphyl to Work with Genetic Data. In *Proc. of the 2003 IEEE Congress on Evolutionary Computation*, pages 320–326. IEEE Computer Society.

[43] Costanza, J., Carapezza, G., Angione, C., Li, P., and Nicosia, G. (2012). Robust design of microbial strains. *Bioinformatics*, 28(23):3097–3104.

[44] Cotta, C. and Moscato, P. (2002). Inferring Phylogenetic Trees Using Evolutionary Algorithms. In *Parallel Problem Solving From Nature VII*, volume 2439 of *LNCS*, pages 720–729. Springer-Verlag.

[45] Cotta, C. and Moscato, P. (2003). A memetic-aided approach to hierarchical clustering from distance matrices: application to gene expression clustering and phylogeny. *Biosystems*, 72(1-2):75–97.

[46] Cracraft, J. and Donoghue, M. J. (2004). *Assembling the Tree of Life*. Oxford University Press, New York, NY.

[47] Cutello, V., Narzisi, G., and Nicosia, G. (2006). A Multi-Objective Evolutionary Approach to the Protein Structure Prediction Problem. *Journal of The Royal Society Interface*, 3(6):139–151.

[48] Darriba, D., Taboada, G. L., Doallo, R., and Posada, D. (2012). jModelTest 2: more models, new heuristics and parallel computing. *Nature Methods*, 9(8):772–772.

[49] Day, R. O. and Lamont, G. B. (2006). Parallel Evolutionary Computations in Discerning Protein Structures. In *Parallel Computing for Bioinformatics and Computational Biology: Models, Enabling Technologies, and Case Studies*, pages 459–486. John Wiley & Sons Inc.

[50] Day, R. O., Zydallis, J. B., Lamont, G. B., and Pachter, R. (2002). Solving the Protein Structure Prediction Problem through a Multiobjective Genetic Algorithm. In *Proc. of the 2002 International Conference on Computational Nanoscience and Nanotechnology*, pages 32–35.

[51] Day, W. H. E., Johnson, D. S., and Sankoff, D. (1986). The Computational Complexity of Inferring Rooted Phylogenies by Parsimony. *Mathematical Biosciences*, 81(1):33–42.

[52] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A Fast and Elitist Multi-Objective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

[53] Díaz, J., Muñoz-Caro, C., and Niño, A. (2012). A Survey of Parallel Programming Models and Tools in the Multi and Many-core Era. *IEEE Transactions on Parallel and Distributed Systems*, 23(8):1369–1386.

[54] Durillo, J. J., Nebro, A. J., Luna, F., and Alba, E. (2008). A Study of Master-Slave Approaches to Parallelize NSGA-II. In *Proc. of the 2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8. IEEE Computer Society.

[55] Dutheil, J. and Boussau, B. (2008). Non-homogeneous models of sequence evolution in the Bio++ suite of libraries and programs. *BMC Evolutionary Biology*, 8(255):1–12.

[56] Edgeworth, F. Y. (1881). *Mathematical Psychics: An Essay on the Application of Mathematics to the Moral Sciences*. C. Kegan Paul and Co., London, United Kingdom.

[57] Felsenstein, J. (1981). Evolutionary Trees from DNA Sequences: A Maximum Likelihood Approach. *Journal of Molecular Evolution*, 17(6):368–376.

[58] Felsenstein, J. (2000). PHYLIP (phylogeny inference package). http://evolution.genetics.washington.edu/phylip.html.

[59] Felsenstein, J. (2004). *Inferring Phylogenies*. Sinauer Associates, Sunderland, Massachusetts.

[60] Ficici, S. G., Liu, E., and Fogel, G. B. (2012). Evolutionary algorithms for supertree search. In *Proc. of the 2012 IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE Computer Society.

[61] Fitch, W. (1972). Toward Defining the Course of Evolution: Minimum Change for a Specific Tree Topology. *Systematic Zoology*, 20(4):406–416.

[62] Fitch, W. and Margoliash, E. (1967). Construction of phylogenetic trees. *Science*, 155(3760):279–284.

[63] Flouri, T., Izquierdo-Carrasco, F., Darriba, D., Aberer, A., Nguyen, L. T., Minh, B. Q., von Haeseler, A., and Stamatakis, A. (2015). The Phylogenetic Likelihood Library. *Systematic Biology*, 64(2):356–362.

[64] Fogel, G. B. (2005). Evolutionary Computation for the Inference of Natural Evolutionary Histories. *IEEE Connections*, 3(1):11–14.

[65] Galindo, I., Almeida, F., and Badia-Contelles, J. M. (2008). Dynamic Load Balancing on Dedicated Heterogeneous Systems. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface, EuroPVM/MPI 2008*, volume 5205 of *LNCS*, pages 64–74. Springer-Verlag.

[66] Gallardo, J. E., Cotta, C., and Fernández, A. J. (2007). Reconstructing Phylogenies with Memetic Algorithms and Branch-and-Bound. In *Analysis of Biological Data: A Soft Computing Approach*, pages 59–84. World Scientific.

[67] Gascuel, O. (1997). BIONJ: An improved version of the NJ algorithm based on a simple model of sequence data. *Molecular Biology and Evolution*, 14(7):685–695.

[68] Gaster, B., Howes, L., Kaeli, D. R., Mistry, P., and Schaa, D. (2012). *Heterogeneous Computing with OpenCL - Revised OpenCL 1.2 Edition*. Morgan Kaufmann Publishers, San Francisco, California.

[69] Glaskowsky, P. N. (2009). NVIDIA's Fermi: The First Complete GPU Computing Architecture.

[70] Goëffon, A., Richer, J. M., and Hao, J. K. (2008). Progressive Tree Neighborhood Applied to the Maximum Parsimony Problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(1):136–145.

[71] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Boston, Massachusetts.

[72] Goloboff, P. A., Farris, J. S., and Nixon, K. C. (2008). TNT, a free program for phylogenetic analysis. *Cladistics*, 24(5):774–786.

[73] Gordon, V. S., Whitley, L. D., and Bohm, A. P. W. (1992). Dataflow Parallelism in Genetic Algorithms. In *Parallel Problem Solving from Nature 2*, pages 539–548. Elsevier.

[74] Gropp, W., Lusk, W., and Skjellum, A. (2014). *Using MPI: Portable Parallel Programming with the Message Passing Interface. 3rd edition*. The MIT Press, Cambridge, Massachusetts.

[75] Guéquen, L. et al. (2013). Bio++: efficient extensible libraries and tools for computational molecular evolution. *Molecular Biology and Evolution*, 30(8):1745–1750.

[76] Guo, H., Lu, Q., Wu, J., Huang, X., and Qian, P. (2009). Solving 2D HP Protein Folding Problem by Parallel Ant Colonies. In *Proc. of the 2nd International Conference on BioMedical Engineering and Informatics*, pages 1–5.

[77] Handl, J., Kell, D. B., and Knowles, J. D. (2007). Multiobjective Optimization in Bioinformatics and Computational Biology. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(2):279–292.

[78] Hasegawa, M., Kishino, H., and Yano, T. (1985). Dating of the human-ape splitting by a molecular clock of mitochondrial DNA. *Journal of Molecular Evolution*, 22(2):160–174.

[79] Hecker, M., Lambeck, S., Toepfer, S., van Someren, E., and Guthke, R. (2009). Gene regulatory network inference: Data integration in dynamic models - A review. *Biosystems*, 96(1):86–103.

[80] Helaers, R. and Milinkovitch, M. C. (2010). MetaPIGA v2.0: maximum likelihood large phylogeny estimation using the metapopulation genetic algorithm and other stochastic heuristics. *BMC Bioinformatics*, 11(1):379–389.

[81] Hennessy, J. L. and Patterson, D. A. (2011). *Computer Architecture: A Quantitative Approach, Fifth Edition*. Morgan Kaufmann Publishers Inc., San Francisco, California.

[82] Hero, A. O., Fleury, G., Mears, A. J., and Swaroop, A. (2004). Multicriteria Gene Screening for Analysis of Differential Expression with DNA Microarrays. *EURASIP Journal on Applied Signal Processing*, 2004(1):43–52.

[83] Hongwei, X. and Yanhua, L. (2009). Parallel ACO for DNA Sequencing by Hybridization. In *Proc. of the 2009 WRI World Congress on Computer Science and Information Engineering*, pages 602–606. IEEE Computer Society.

[84] Huang, W. and Tafti, D. K. A. (1999). A Parallel Computing Framework for Dynamic Power Balancing in Adaptive Mesh Refinement Applications. In *Parallel Computational Fluid Dynamics '99: Towards Teraflops, Optimization and Novel Formulations*, pages 249–256.

[85] Husník, F., Chrudimský, T., and Hypša, V. (2011). Multiple origins of endosymbiosis within the Enterobacteriaceae (γ-Proteobacteria): convergence of complex phylogenetic approaches. *BMC Biology*, 9(87):1–17.

[86] Huson, D. H. and Bryant, D. (2006). Application of Phylogenetic Networks in Evolutionary Studies. *Molecular Biology and Evolution*, 23(2):254–267.

[87] Ingman, M. and Gyllensten, U. (2006). mtDB: Human Mitochondrial Genome Database, a resource for population genetics and medical sciences. *Nucleic Acids Research*, 34 (Database issue D749-D751).

[88] Intel Corporation (2011). Writing Optimal OpenCL Code with Intel OpenCL SDK.

[89] Intel Corporation (2014). OpenCL Device Fission for CPU Performance.

[90] Islam, R. and Ngom, A. (2006). Protein Threading using Parallel Evolution Strategy. In *Proc. of the 2006 IEEE Congress on Evolutionary Computation*, pages 2347–2354. IEEE Computer Society.

[91] Izquierdo-Carrasco, F., Alachiotis, N., Berger, S., Flouri, T., Pissis, S. P., and Stamatakis, A. (2013). A Generic Vectorization Scheme and a GPU Kernel for the Phylogenetic Likelihood Library. In *Proc. of the 27th IEEE International Parallel & Distributed Processing Symposium*, pages 530–538. IEEE Computer Society.

[92] Jakobović, D., Golub, M., and Čupić, M. (2014). Asynchronous and implicitly parallel evolutionary computation models. *Soft Computing*, 18(6):1225–1236.

[93] Jones, M. D. and Yao, R. (2006). Hybrid MPI-OpenMP Programming for Parallel OSEM PET Reconstruction. *IEEE Transactions on Nuclear Science*, 53(5):2752–2758.

[94] Jukes, T. H. and Cantor, C. R. (1969). Evolution of protein molecules. In *Mammalian Protein Metabolism Vol. III*, pages 21–132. Academic Press.

[95] Karaboga, D. and Basturk, B. (2007). A Powerful and Efficient Algorithm for Numerical Function Optimization: Artificial Bee Colony (ABC) Algorithm. *Journal of Global Optimization*, 39(3):459–171.

[96] Katoh, K., Kuma, K., and Miyata, T. (2001). Genetic algorithm-based maximum-likelihood analysis for molecular phylogeny. *Journal of Molecular Evolution*, 53(4-5):477–484.

[97] Khabzaoui, M., Dhaenens, C., and Talbi, E. G. (2004). A Multicriteria Genetic Algorithm to Analyze Microarray Data. In *Proc. of the 2004 IEEE Congress on Evolutionary Computation*, pages 1874–1881. IEEE Computer Society.

[98] Khabzaoui, M., Dhaenens, C., and Talbi, E. G. (2006). A cooperative genetic algorithm for knowledge discovery in microarray experiments. In *Parallel Computing for Bioinformatics and Computational Biology: Models, Enabling Technologies, and Case Studies*, pages 303–324. John Wiley & Sons Inc.

[99] King, B. (1967). Step-wise clustering procedures. *Journal of the American Statistical Association*, 62(317):86–101.

[100] Kishino, H. and Hasegawa, M. (1989). Evaluation of the maximum likelihood estimate of the evolutionary tree topologies from DNA sequence data. *Journal of Molecular Evolution*, 29(2):170–179.

[101] Koduru, P., Das, S., Welch, S., Roe, J. L., and Lopez-Dee, Z. P. (2005). A Co-evolutionary Hybrid Algorithm for Multi-Objective Optimization of Gene Regulatory Network Models. In *Genetic and Evolutionary Computation Conference*, pages 393–399.

[102] Kuan, L., Neves, J., Pratas, F., Tomás, P., and Sousa, L. (2014). Accelerating Phylogenetic Inference on GPUs: an OpenACC and CUDA Comparison. In *Proc. of the 2nd International Work-Conference on Bioinformatics and Biomedical Engineering*, pages 589–600.

[103] Lastovetsky, A. L. and Dongarra, J. J. (2009). Distribution of Computations with Constant Performance Models of Heterogeneous Processors. In *High-Performance Heterogeneous Computing*, pages 25–59. John Wiley & Sons Inc.

[104] Lee, W. P., Hsiao, Y. T., and Hwang, W. C. (2014). Designing a parallel evolutionary algorithm for inferring gene networks on the cloud computing environment. *BMC Systems Biology*, 8(5):1–19.

[105] Lemey, P., Salemi, M., and Vandamme, A.-M. (2009). *The Phylogenetic Handbook: a Practical Approach to Phylogenetic Analysis and Hypothesis Testing*. Cambridge Univ. Press, Cambridge.

[106] Lemmon, A. R. and Milinkovitch, M. C. (2002). The metapopulation genetic algorithm: An efficient solution for the problem of large phylogeny estimation. *Proc. Natl. Acad. Sci. U.S.A.*, 99(16):10516–10521.

[107] Lewis, P. O. (1998). A Genetic Algorithm for Maximum-Likelihood Phylogeny Inference Using Nucleotide Sequence Data. *Molecular Biology and Evolution*, 15(3):277–283.

[108] Loft, R. D., Thomas, S. J., and Dennis, J. M. (2001). Terascale Spectral Element Dynamical Core for Atmospheric General Circulation Models. In *Proc. of the ACM/IEEE 2001 Conference on Supercomputing*, pages 1–32.

[109] Luong, P., Breshears, C. P., and Ly, L. N. (2001). Costal Ocean Modeling of the U.S. West Coast with Multiblock Grid and Dual-Level Parallelism. In *Proc. of the ACM/IEEE 2001 Conference on Supercomputing*, pages 1–18.

[110] Luque, G., Alba, E., and Khuri, S. (2006). Assembling DNA Fragments with a Distributed Genetic Algorithm. In *Parallel Computing for Bioinformatics and Computational Biology: Models, Enabling Technologies, and Case Studies*, pages 285–302. John Wiley & Sons Inc.

[111] Macey, J. R. (2005). Plethodontid salamander mitochondrial genomics: A parsimony evaluation of character conflict and implications for historical biogeography. *Cladistics*, 21(2):194–202.

[112] Mahoney, A. W., Podgorski, G. J., and Flann, N. S. (2012). Multiobjective Optimization Based-Approach for Discovering Novel Cancer Therapies. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(1):169–284.

[113] Märtens, M. and Izzo, D. (2013). The Asynchronous Island Model and NSGA-II: Study of a New Migration Operator and its Performance. In *Genetic and Evolutionary Computation Conference*, pages 1173–1180.

[114] Matsuda, H. (1995). Construction of Phylogenetic Trees from Amino acid Sequences using a Genetic Algorithm. In *Genome Informatics Workshop*, pages 19–28. Universal Academy Press.

[115] Mauro, D. S., Gower, D. J., Müller, H., Loader, S. P., Zardoya, R., Nussbaum, R. A., and Wilkinson, M. (2014). Life-history evolution and mitogenomic phylogeny of caecilian amphibians. *Molecular Phylogenetics and Evolution*, 73(1):177–189.

[116] Meier-Kolthoff, J., Auch, A., Klenk, H. P., and Göker, M. (2014). Highly Parallelized Inference of Large Genome-Based Phylogenies. *Concurrency and Computation: Practice and Experience*, 26(10):1715–1729.

[117] Meredith, R. W. et al. (2011). Impacts of the Cretaceous Terrestrial Revolution and KPg Extinction on Mammal Diversification. *Science*, 334(6055):521–524.

[118] Millonas, M. M. (1994). Swarms, Phase Transitions, and Collective Intelligence. In *Artificial Life III*, pages 417–445. Addison-Wesley, Reading.

[119] Minh, B. Q., Vinh, L. S., Schmidt, H. A., and von Haeseler, A. (2006). Large Maximum Likelihood Trees. In *Proc. of the NIC Symposium*, pages 357–366. Forschungszentrum Jülich, Germany.

[120] Minh, B. Q., Vinh, L. S., von Haeseler, A., and Schmidt, H. A. (2005). pIQPNNI - parallel reconstruction of large maximum likelihood phylogenies. *Bioinformatics*, 21(19):3794–3796.

[121] Moilanen, A. (1999). Searching for Most Parsimonious Trees with Simulated Evolutionary Optimization. *Cladistics*, 15(1):39–50.

[122] Moore, M. J. and Jansen, R. K. (2006). Molecular evidence for the age, origin, and evolutionary history of the american desert plant genus tiquilia (boraginaceae). *Molecular Phylogenetics and Evolution*, 39(3):668–687.

[123] Moret, B. M. E., Bader, D. A., and Warnow, T. (2002). High-Performance Algorithm Engineering for Computational Phylogenetics. *The Journal of Supercomputing*, 22(1):99–111.

[124] Mueller, R. L., Macey, J. R., Jaekel, M., Wake, D. B., and Boore, J. L. (2004). Morphological homoplasy, life history evolution, and historical biogeography of plethodontid salamanders inferred from complete mitochondrial genomes. *Proc. Natl. Acad. Sci. U.S.A.*, 101(38):13820–13825.

[125] NVIDIA Corporation (2011). OpenCL Best Practices Guide.

[126] NVIDIA Corporation (2012). OpenCL Programming for the CUDA Architecture.

[127] Ocaña, K. A. C. S., Oliveira, D., Dias, J., and Ogasawara, E. (2013). Designing a parallel cloud based comparative genomics workflow to improve phylogenetic analyses. *Future Generation Computer Systems*, 29(8):2205–2219.

[128] Ogden, T. H. and Rosenberg, M. S. (2007). How should gaps be treated in parsimony? A comparison of approaches using simulation. *Molecular Phylogenetics and Evolution*, 42(3):817–826.

[129] Ortega, J., Anguita, M., and Prieto, A. (2005). *Arquitectura de Computadores*. Thomson, Madrid, Spain.

[130] Ortuño, F., Florido, J. P., Urquiza, J. M., Pomares, H., Prieto, A., and Rojas, I. (2012). Optimization of multiple sequence alignment methodologies using a multiobjective evolutionary algorithm based on NSGA-II. In *Proc. of the 2012 IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE Computer Society.

[131] Ott, M., Zola, J., Aluru, S., Johnson, A. D., Janies, D., and Stamatakis, A. (2008). Large-scale phylogenetic analysis on current HPC architectures. *Scientific Programming*, 16(2-3):255–270.

[132] Pal, S. K., Bandyopadhyay, S., and Ray, S. S. (2006). Evolutionary Computation in Bioinformatics: A Review. *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, 36(5):601–615.

[133] Pareto, V. (1896). *Cours D'Economie Politique*. Rouge, Lausanne, Switzerland.

[134] Pavlopoulou, A. and Michalopoulos, I. (2011). State-of-the-art bioinformatics protein structure prediction tools (Review). *International Journal of Molecular Medicine*, 28(3):295–310.

[135] Penas, D. R., Banga, J. R., González, P., and Doallo, R. (2014). 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 of *Advances in Intelligent Systems and Computing*, pages 173–181. Springer.

[136] Pfeiffer, W. and Stamatakis, A. (2010). Hybrid MPI/Pthreads Parallelization of the RAxML Phylogenetics Code. In *Proc. of the Ninth IEEE International Workshop on High Performance Computational Biology*, pages 1–8. IEEE Computer Society.

[137] Poladian, L. (2005). A GA for maximum likelihood phylogenetic inference using neighbour-joining as a genotype to phenotype mapping. In *Genetic and Evolutionary Computation Conference*, pages 415–422.

[138] Poladian, L. and Jermiin, L. (2006). Multi-Objective Evolutionary Algorithms and Phylogenetic Inference with Multiple Data Sets. *Soft Computing*, 10(4):359–368.

[139] Pratas, F., Trancoso, P., Sousa, L., Stamatakis, A., Shi, G., and Kindratenko, V. (2012). Fine-grain parallelism using multi-core, Cell/BE, and GPU systems. *Parallel Computing*, 38(8):365–390.

[140] Rausch, T., Thomas, A., Camp, N., Cannon, L., and Facelli, J. (2008). A Parallel Genetic Algorithm to Discover Patterns in Genetic Markers that Indicate Predisposition to Multifactorial Disease. *Computers in Biology and Medicine*, 38(7):826–836.

[141] Reijmers, T. H., Wehrens, R., Daeyaert, F. D., Lewi, P. J., and Buydens, L. M. C. (1999). Using genetic algorithms for the construction of phylogenetic trees: application to G-protein coupled receptor sequences. *Biosystems*, 49(1):31–43.

[142] Ribeiro, C. C. and Vianna, D. S. (2009). A hybrid genetic algorithm for the phylogeny problem using path-relinking as a progressive crossover strategy. *International Transactions in Operational Research*, 16(5):641–657.

[143] Richer, J. M., Goëffon, A., and Hao, J. K. (2009). A Memetic Algorithm for Phylogenetic Reconstruction with Maximum Parsimony. In *Evolutionary Computation, Machine Learning and Data Mining in Computational Biology*, volume 5483 of *LNCS*, pages 164–175. Springer-Verlag.

[144] Richer, J. M., Rodriguez-Tello, E., and Vazquez-Ortiz, K. E. (2013). Maximum Parsimony Phylogenetic Inference Using Simulated Annealing. In *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation II*, volume 175 of *Advances in Intelligent Systems and Computing*, pages 189–203. Springer-Verlag.

[145] Rodriguez, F., Oliver, J. F., Marin, A., and Medina, J. R. (1990). The general stochastic model of nucleotide substitution. *Journal of Theoretical Biology*, 142(4):485–501.

[146] Rokas, A., Williams, B. L., King, N., and Carroll, S. B. (2003). Genome-scale approaches to resolving incongruence in molecular phylogenies. *Nature*, 425(6960):798–804.

[147] Ronquist, F. (1998). Fast Fitch-Parsimony Algorithms for Large Data Sets. *Cladistics*, 14(4):387–400.

[148] Ronquist, F. et al. (2012). MrBayes 3.2: Efficient Bayesian Phylogenetic Inference and Model Choice Across a Large Model Space. *Systematic Biology*, 61(3):539–542.

[149] Saitou, N. and Nei, M. (1987). The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425.

[150] Sarkar, S., Majumder, T., Kalyanaraman, A., and Pande, P. P. (2010). Hardware Accelerators for Biocomputing: A Survey. In *Proc. of the 2010 IEEE International Symposium on Circuits and Systems*, pages 3789–3792. IEEE Computer Society.

[151] Schmidt, H. A., Petzold, E., Vingron, M., and von Haeseler, A. (2003). Molecular phylogenetics: parallelized parameter estimation and quartet puzzling. *Journal of Parallel and Distributed Computing*, 63(7-8):719–727.

[152] Schmidt, O., Drake, H. L., and Horn, M. A. (2010). Hitherto Unknown [Fe-Fe]-Hydrogenase Gene Diversity in Anaerobes and Anoxic Enrichments from a Moderately Acidic Fen. *Applied and Environmental Microbiology*, 76(6):2027–2031.

[153] Seeley, T. D. (1995). *The Wisdom of the Hive*. Harvard University Press, Cambridge, Massachusetts.

[154] Shapiro, B. A., Wu, J. C., Bengali, D., and Potts, M. J. (2001). The massively parallel genetic algorithm for RNA folding: MIMD implementation and population variation. *Bioinformatics*, 17(2):137–148.

[155] Shen, J. and Heckendorn, R. B. (2004). Discrete Branch Length Representations for Genetic Algorithms in Phylogenetic Search. In *Applications of Evolutionary Computing*, volume 3005 of *LNCS*, pages 94–103. Springer-Verlag.

[156] Shendure, J. and Ji, H. (2008). Next-generation DNA sequencing. *Nature Biotechnology*, 26(1):1135–1145.

[157] Sheskin, D. J. (2011). *Handbook of Parametric and Nonparametric Statistical Procedures. 5th edition*. Chapman & Hall/CRC Press, NY, USA.

[158] Shimodaira, H. and Hasegawa, M. (1999). Multiple comparisons of log-likelihoods with applications to phylogenetic inference. *Molecular Biology and Evolution*, 16(8):1114–1116.

[159] Shimodaira, H. and Hasegawa, M. (2001). CONSEL: for assessing the confidence of phylogenetic tree selection. *Bioinformatics*, 17(12):1246–1247.

[160] Shin, S. Y., Lee, I. H., Kim, D., and Zhang, B. T. (2005). Multiobjective Evolutionary Optimization of DNA Sequences for Reliable DNA Computing. *IEEE Transactions on Evolutionary Computation*, 9(2):143–158.

[161] Sindhya, K., Sinha, A., Deb, K., and Miettinen, K. (2009). Local Search Based Evolutionary Multi-Objective Optimization Algorithm for Constrained and Unconstrained Problems. In *Proc. of the 2009 IEEE Congress on Evolutionary Computation*, pages 2919–2926. IEEE Computer Society.

[162] Skourikhine, A. (2000). Phylogenetic Tree Reconstruction Using Self-Adaptive Genetic Algorithm. In *Proc. of the 1st IEEE International Symposium on Bioinformatics and Biomedical Engineering*, pages 129–134. IEEE Computer Society.

[163] Sneath, P. H. A. and Sokal, R. R. (1973). *Numerical Taxonomy: the principles and practice of numerical classification*. W. H. Freeman, San Francisco, California.

[164] Snell, Q., Whiting, M., Clement, M., and McLaughlin, D. (2000). Parallel phylogenetic inference. In *Proc. of the 2000 ACM/IEEE conference on Supercomputing*, page Article 35. IEEE Computer Society.

[165] Sokal, R. R. and Michener, C. D. (1958). A statistical method for evaluating systematic relationships. *Universtiy of Kansas Science Bulletin*, 38(2):1409–1438.

[166] Song, F., Tomov, S., and Dongarra, J. (2012). Enabling and scaling matrix computations on heterogeneous multi-core and multi-GPU systems. In *Proc. of the 26th ACM International Conference on Supercomputing*, pages 365–376. Association for Computing Machinery (ACM).

[167] Sperschneider, V. (2008). *Bioinformatics - Problem Solving Paradigms*. Springer-Verlag, Berlin Heidelberg.

[168] Spieth, C., Streichert, F., Speer, N., and Zell, A. (2005). Multi-Objective Model Optimization for Inferring Gene Regulatory Networks. In *Evolutionary Multi-Criterion Optimization*, volume 3410 of *LNCS*, pages 607–620. Springer-Verlag.

[169] Stamatakis, A. (2005a). An Efficient Program for Phylogenetic Inference Using Simulated Annealing. In *Proc. of the 19th IEEE International Parallel and Distributed Processing Symposium*, pages 1–8. IEEE Computer Society.

[170] Stamatakis, A. (2005b). Phylogenetics: Applications, Software and Challenges. *Cancer Genomics and Proteomics*, 2(5):301–305.

[171] Stamatakis, A. (2014). RAxML Version 8: A Tool for Phylogenetic Analysis and Post-Analysis of Large Phylogenies. *Bioinformatics*, 30(9):1312–1313.

[172] Stamatakis, A. and Ott, M. (2008). Exploiting Fine-Grained Parallelism in the Phylogenetic Likelihood Function with MPI, Pthreads, and OpenMP: A Performance Study. In *Pattern Recognition in Bioinformatics*, volume 5265 of *LNBI*, pages 424–436. Springer.

[173] Stewart, C., Hart, D., Berry, D., Olsen, G., Wernert, E., and Fischer, W. (2001). Parallel implementation and performance of fastDNAml: a program for maximum likelihood phylogenetic inference. In *Proc. of the 14th IEEE/ACM Supercomputing Conference*, pages 20–20. Association for Computing Machinery (ACM).

[174] Suchard, M. A. and Rambaut, A. (2009). Many-core algorithms for statistical phylogenetics. *Bioinformatics*, 25(11):1370–1376.

[175] Swofford, D. L., Waddell, P. J., Huelsenbeck, J. P., Foster, P. G., Lewis, P. O., and Rogers, J. S. (2001). Bias in phylogenetic estimation and its relevance to the choice between parsimony and likelihood methods. *Systematic Biology*, 50(4):525–539.

[176] Talbi, E. G. (2009). *Metaheuristics: From Design to Implementation*. John Wiley & Sons Inc., Hoboken, New Jersey.

[177] Talbi, E. G., Mostaghim, S., Okabe, T., Ishibuchi, H., and Coello, C. (2008). Parallel Approaches for Multiobjective Optimization. In *Multiobjective Optimization*, volume 5252 of *LNCS*, pages 349–372. Springer-Verlag.

[178] Tamura, K. and Nei, M. (1993). Estimation of the number of nucleotide substitutions in the control region of mitochondrial dna in humans and chimpanzees. *Molecular Biology and Evolution*, 10(3):512–526.

[179] Taneda, A. (2010). Multi-objective pairwise RNA sequence alignment. *Bioinformatics*, 26(19):2383–2390.

[180] Tantar, A., Melab, N., Talbi, E. G., Parent, B., and Horvath, D. (2007). A parallel hybrid genetic algorithm for protein structure prediction on the computational grid. *Future Generation Computer Systems*, 23(3):398–409.

[181] Teodoro, G., Oliveira, R. S., Sertel, O., Gurcan, M. N., Meira, W., Catalyurek, U., and Ferreira, R. (2009). Coordinating the use of GPU and CPU for improving performance of compute intensive applications. In *Proc. of the 2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–10. IEEE Computer Society.

[182] Timme, R. E., Pettengill, J. B., Allard, M. W., Strain, E., Barrangou, R., Wehnes, C., Kessel, J. S. V., Karns, J. S., Musser, S. M., and Brown, E. W. (2013). Phylogenetic diversity of the enteric pathogen Salmonella enterica subsp. enterica inferred from genome-wide reference-free SNP characters. *Genome Biology and Evolution*, 5(11):2109–2123.

[183] Veldhuizen, D. A. V. and Lamont, G. B. (2000). On Measuring Multiobjective Evolutionary Algorithm Performance. In *Proc. of the 2000 IEEE Congress on Evolutionary Computation*, pages 204–211. IEEE Computer Society.

[184] Veldhuizen, D. A. V., Zydallis, J. B., and Lamont, G. B. (2003). Considerations in Engineering Parallel Multiobjective Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 7(2):144–173.

[185] Vieites, D. R., Román, S. N., Wake, M. H., and Wake, D. B. (2011). A multigenic perspective on phylogenetic relationships in the largest family of salamanders, the Plethodontidae. *Molecular Phylogenetics and Evolution*, 59(3):623–635.

[186] Weber, R., Gothandaraman, A., Hinde, R. J., and Peterson, G. D. (2011). Comparing Hardware Accelerators in Scientific Applications: A Case Study. *IEEE Transactions on Parallel and Distributed Systems*, 22(1):58–68.

[187] Wilt, N. (2013). *The CUDA Handbook: A Comprehensive Guide to GPU Programming*. Addison-Wesley, Upper Saddle River, New Jersey.

[188] Wirawan, A., Keong, K., and Schmidt, B. (2008). Parallel DNA sequence alignment on the cell broadband engine. In *Parallel Processing and Applied Mathematics*, volume 4967 of *LNCS*, pages 1249–1256. Springer Berlin.

[189] Yagoubi, M. and Schoenauer, M. (2012). Asynchronous Master/Slave MOEAs and Heterogeneous Evaluation Costs. In *Genetic and Evolutionary Computation Conference*, pages 1007–1014.

[190] Yang, X.-S. (2010). Firefly algorithm, Stochastic Test Functions and Design Optimisation. *International Journal of Bio-Inspired Computation*, 2(2):78–84.

[191] Yang, X.-S. (2013). Bat Algorithm: Literature Review and Applications. *International Journal of Bio-Inspired Computation*, 5(3):141–149.

[192] Zhang, Q. and Li, H. (2007). MOEA/D: A Multi-objective Evolutionary Algorithm Based on Decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731.

[193] Zhang, Q., Zhao, A., Suganthan, P. N., Liu, W., and Tiwari, S. (2008). Multiobjective optimization test instances for the CEC09 special session and competition. *Technical Report CES-487, University of Essex and Nanyang Technological University*.

[194] Zitzler, E., Deb, K., and Thiele, L. (2000). Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2):173–195.

[195] Zitzler, E. and Künzli, S. (2004). Indicator-Based Selection in Multiobjective Search. In *Parallel Problem Solving From Nature VIII*, volume 3242 of *LNCS*, pages 832–842. Springer-Verlag.

[196] Zitzler, E., Laumanns, M., and Thiele, L. (2002). SPEA2: Improving the Strength Pareto Evolutionary Algorithm. In *Proceedings of Evolutionary Methods for Design, Optimization, and Control with Applications to Industrial Problems (EUROGEN'02)*, pages 95–100.

[197] Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., and Fonseca, V. G. D. (2003). Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132.

[198] Zomaya, A. Y. (2006). *Parallel Computing for Bioinformatics and Computational Biology: Models, Enabling Technologies, and Case Studies*. John Wiley & Sons Inc., Hoboken, New Jersey.

[199] Zăvoianu, A., Lughofer, E., Koppelstätter, W., Weidenholzer, G., Amrhein, W., and Klement, E. P. (2013). On the Performance of Master-Slave Parallelization Methods for Multi-Objective Evolutionary Algorithms. In *Artificial Intelligence and Soft Computing*, volume 7895 of *LNCS*, pages 122–134. Springer-Verlag.

[200] Zwickl, D. (2006). *Genetic Algorithm Approaches for the Phylogenetic Analysis of Large Biological Sequence Datasets Under the Maximum Likelihood Criterion. Ph.D. Thesis*. Univ. Texas at Austin, USA.