PEDECIBA INFORMÁTICA

Instituto de Computación, Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

TESIS DE DOCTORADO EN INFORMÁTICA

Parallel evolutionary algorithms for scheduling on heterogeneous computing and grid environments

Sergio Nesmachnow
sergion@fing.edu.uy

Abril de 2010

Supervisores: Enrique Alba, Héctor Cancela

Tribunal: Celso Ribeiro, El-Ghazali Talbi (revisores),
Irene Loiseau, Alberto Pardo, María Urquhart

Parallel evolutionary algorithms for scheduling on heterogeneous computing and grid environments
Nesmachnow, Sergio
ISSN 0797-6410
Tesis de Doctorado en Informática
Reporte Técnico RT 10-05
PEDECIBA
Instituto de Computación - Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay, abril de 2010

Parallel evolutionary algorithms for scheduling on heterogeneous computing and grid environments

Abstract

This thesis studies the application of sequential and parallel evolutionary algorithms to the scheduling problem in heterogeneous computing and grid environments, a key problem when executing tasks in distributed computing systems. Since the 1990's, this class of systems has been increasingly employed to provide support for solving complex problems using high-performance computing techniques. The scheduling problem in heterogeneous computing systems is an NP-hard optimization problem, which has been tackled using several optimization methods in the past. Among many new techniques for optimization, evolutionary computing methods have been successfully applied to this class of problems. In this work, several evolutionary algorithms in their sequential and parallel variants are specifically designed to provide accurate solutions for the problem, allowing to compute an efficient planning for heterogeneous computing and grid environments. New problem instances, far more complex than those existing in the related literature, are introduced in this thesis in order to study the scalability of the presented parallel evolutionary algorithms. In addition, a new parallel micro-CHC algorithm is developed, inspired by useful ideas from the multiobjective optimization field. Efficient numerical results of this algorithm are reported in the experimental analysis performed on both well-known problem instances and the large instances specially designed in this work. The comparative study including traditional methods and evolutionary algorithms shows that the new parallel micro-CHC is able to achieve a high problem solving efficacy, outperforming previous results already reported for the problem and also having a good scalability behavior when solving high dimension problem instances. In addition, two variants of the scheduling problem in heterogeneous environments are also tackled, showing the versatility of the proposed approach using parallel evolutionary algorithms to deal with both dynamic and multiobjective scenarios.

Keywords: Parallel evolutionary algorithms, Scheduling, Heterogeneous computing, Grid computing

Algoritmos evolutivos paralelos para planificación de tareas en entornos heterogéneos

Resumen

Esta tesis estudia la aplicación de algoritmos evolutivos secuenciales y paralelos para el problema de planificación de tareas en entornos de cómputo heterogéneos y de computación grid. Desde la década de 1990, estos sistemas computacionales han sido utilizados con éxito para resolver problemas complejos utilizando técnicas de computación de alto desempeo. El problema de planificación de tareas en entornos heterogéneos es un problema de optimización NP-difícil que ha sido abordado utilizando diversas técnicas. Entre las técnicas emergentes para optimización combinatoria, los algoritmos evolutivos han sido aplicados con éxito a esta clase de problemas. En este trabajo, varios algoritmos evolutivos en sus versiones secuenciales y paralelas han sido específicamente diseados para alcanzar soluciones precisas para el problema de planificación de tareas en entornos de heterogéneos, permitiendo calcular planificaciones eficientes para entornos que modelan clusters de computadores y plataformas de computación grid. Nuevas instancias del problema, con una complejidad mucho mayor que las previamente existentes en la literatura relacionada, son presentadas en esta tesis con el objetivo de analizar la escalabilidad de los algoritmos evolutivos propuestos. Complementariamente, un nuevo método, el micro-CHC paralelo es desarrollado, inspirado en ideas útiles provenientes del área de optimización multiobjetivo. Resultados numéricos precisos y eficientes se reportan en el análisis experimental realizado sobre instancias estándar del problema y sobre las nuevas instancias específicamente diseadas en este trabajo. El estudio comparativo que incluye a métodos tradicionales para planificación de tareas, los nuevos métodos propuestos y algoritmos evolutivos previamente aplicados al problema, demuestra que el nuevo micro-CHC paralelo es capaz de alcanzar altos valores de eficacio, superando a los mejores resultados previamente reportados en la literatura del área y mostrando un buen comportamiento de escalabilidad para resolver las instancias de gran dimensión. Además, dos variantes del problema de planificación de tareas en entornos heterogéneos han sido inicialmente estudiadas, comprobándose la versatilidad del enfoque propuesto para resolver las variantes dinámica y multiobjetivo del problema.

Palabras clave: Algoritmos evolutivos paralelos, Planificación de tareas, Computación heterogénea, Computación grid

Contents

1 Introduction **3**

2 Parallel evolutionary algorithms **7**
2.1 Introduction . . . 7
2.2 Evolutionary algorithms . . . 8
2.3 Genetic algorithm . . . 9
2.4 CHC algorithm . . . 9
2.5 Parallel evolutionary algorithms . . . 10
2.5.1 Master-slave PEAs . . . 11
2.5.2 Cellular PEAs . . . 12
2.5.3 Distributed subpopulations PEAs . . . 12
2.6 Parallel micro-CHC algorithm . . . 14
2.7 Summary . . . 16

3 Scheduling in heterogeneous computing environments **17**
3.1 Heterogeneous computing . . . 17
3.2 HCSP formulation . . . 18
3.2.1 Problem model considerations . . . 20
3.2.2 HCSP computational complexity . . . 20
3.3 Execution time estimation . . . 21
3.3.1 Expected time to compute estimation model . . . 21
3.3.2 Methods for generating ETC scenarios . . . 23
3.4 HCSP instances . . . 23
3.4.1 Instances from Braun et al. (2001) . . . 24
3.4.2 Grid infrastructures . . . 24
3.4.3 New HCSP instances . . . 26
3.5 Static scheduling using execution time estimation . . . 28
3.5.1 Traditional scheduling heuristics . . . 28
3.5.2 Metaheuristics for scheduling using ETC . . . 30
3.6 Summary . . . 33

4 Related work: EAs for HC scheduling **35**
4.1 Introduction . . . 35
4.2 1995-2000: HSCP on multiprocessors . . . 36
4.3 2000-2005: HCSP on distributed environments . . . 38
4.4 2005-2009: Heterogeneous grid scheduling . . . 40
4.5 Summary . . . 42

5 Parallel evolutionary algorithms for the HCSP 45
5.1 The MALLBA library . . . 45
5.2 Problem encoding . . . 47
5.3 Fitness function . . . 48
5.4 Population initialization . . . 48
5.5 Evolutionary operators . . . 49
5.5.1 Exploitation: recombination . . . 49
5.5.2 Exploration: mutation and reinitialization . . . 49
5.6 Local search: randomized PALS . . . 51
5.7 Speeding up the pμ-CHC search . . . 54
5.7.1 Accelerated convergence in pμ-CHC . . . 54
5.7.2 Parallel model of pμ-CHC . . . 55
5.8 Summary . . . 56

6 Experimental analysis 57
6.1 HCSP instances . . . 57
6.2 Development and execution platform . . . 57
6.3 Parameter settings . . . 58
6.3.1 Stopping criterion . . . 58
6.3.2 Population size . . . 59
6.3.3 Operators probabilities . . . 59
6.3.4 Parallel algorithms . . . 60
6.3.5 Probabilities within the evolutionary operators . . . 63
6.3.6 Summary: best parameter configurations . . . 63
6.4 Results and discussion . . . 64
6.4.1 Results for HCSP instances from Braun et al. (2001) . . . 64
6.5 Results for the new large-sized HCSP instances . . . 70
6.5.1 Parallel CHC . . . 70
6.5.2 Parallel micro-CHC . . . 74
6.6 Comparison with lower bounds for the preemptive HCSP . . . 80
6.6.1 Small HCSP instances . . . 82
6.6.2 New large HCSP instances . . . 83
6.6.3 Summary . . . 83
6.7 Tracking the makespan evolution and the execution time . . . 85
6.7.1 Comparative makespan evolution of CHC algorithms . . . 86
6.7.2 Makespan evolution and execution time of pμ-CHC . . . 87
6.8 Scalability analysis and parallel performance . . . 89
6.8.1 Parallel CHC . . . 89
6.8.2 Parallel micro CHC . . . 91
6.9 Summary . . . 92

7 Two HCSP variants: rescheduling and multiobjective approach 93
7.1 Introduction . . . 93
7.2 Dynamic HCSP: rescheduling with pμ-CHC . . . 93
7.2.1 Dynamic HCSP model . . . 93
7.2.2 Rescheduling algorithms . . . 95
7.2.3 Experimental analysis . . . 96
7.2.4 Concluding remarks . . . 98

7.3 Multiobjective HCSP: makespan and flowtime 99
7.3.1 Introduction 99
7.3.2 Multiobjective optimization problems 99
7.3.3 Multiobjective HCSP formulation 100
7.3.4 Multiobjective evolutionary algorithms 101
7.3.5 MOEAs for the HCSP 103
7.3.6 Experimental analysis 104
7.3.7 Concluding remarks 111
7.4 Summary 111

8 Conclusions and future work **113**
8.1 Conclusions 113
8.2 Future work 116

A HCSP instances generator and test suites **119**
A.1 HCSP instances generator 119
A.2 Test suites 124

B Best solutions for the HCSP test suite from Braun et al. (2001) **127**

C Related publications by the author **133**

Bibliography **135**

List of Figures

2.1 Diagram of a master slave PEA. 11
2.2 Diagram of a cellular PEA. 12
2.3 Diagram of a distributed subpopulation PEA. 13

3.1 Scheduling in an HC environment. 18
3.2 Details of the HCSP instances generator. 26
3.3 Format of the HCSP instance files. 27

4.1 Timeline of EAs applied to the HCSP. 36

5.1 UML for MALLBA classes. 46
5.2 Task oriented encoding. 47
5.3 Machine oriented encoding. 48
5.4 Two-level parallel model used in pμ-CHC. 55

6.1 Analysis of the operators probabilities (sequential algorithms, `u_i_hilo.0`). 60
6.2 Analysis of the operators probabilities (parallel algorithms). 60
6.3 Sample results when configuring the number of subpopulations in pCHC. 61
6.4 Sample results when configuring the number of subpopulations in pμ-CHC. 62
6.5 Comparison of pCHC against the best-known methods for the HCSP. . 68
6.6 pCHC improvement over traditional heuristics. 73
6.7 pμ-CHC improvements with respect to deterministic heuristics. 76
6.8 pμ-CHC improvements regarding the consistency classification. 77
6.9 pμ-CHC improvements with respect to pCHC. 79
6.10 HCSP scenario specification in GMPL/AMPL. 81
6.11 Relationship between GAP(optimum) and GAP(LB). 81
6.12 pμ-CHC improvements with respect to the best deterministic heuristic results and the computed lower bounds for HCSP instances by Braun et al. (2001). 83
6.13 pμ-CHC improvements with respect to the best deterministic heuristic result and the computed lower bounds for the preemptive HCSP version. 85
6.14 Makespan evolution for CHC algorithms on `u_i_hihi.0`. 86
6.15 Makespan evolution for CHC algorithms on `u_s_lohi.0`. 86
6.16 Evolution of the makespan improvement ratio for pμ-CHC. 88
6.17 Execution times required to achieve a given improvement threshold. . . 88
6.18 Scalability analysis for pCHC. 89
6.19 Parallel performance and scalability of pCHC. 90
6.20 Scalability analysis for pμ-CHC. 91

6.21 Scalability and parallel performance analysis for pμ-CHC. 92

7.1 Rescheduling in a dynamic scenario. 94
7.2 Pareto fronts for consistent HCSP instances. 108
7.3 Pareto fronts for inconsistent HCSP instances. 109
7.4 Pareto fronts for semiconsistent HCSP instances. 110

List of Tables

3.1 Heterogeneity and consistency combinations in the ETC model. 22
3.2 Parameters of ETC models. 23
3.3 Details of sampled grid and volunteer distributed computing platforms. 25
3.4 Summary of metaheuristic methods applied to solve the HCSP. 30

4.1 EAs applied to the HSCP. 43

6.1 Sample results when configuring the number of subpopulations in pCHC. 61
6.2 Sample results when configuring the number of subpopulations in pμ-CHC. 62
6.3 Best parameter configurations (sequential algorithms). 63
6.4 Best parameter configurations (parallel algorithms). 63
6.5 Results of the sequential EAs for the HCSP. 64
6.6 Comparative results: sequential EAs vs. deterministic heuristics. 65
6.7 Comparative results: sequential EAs vs. previous EAs for the HCSP. . . 65
6.8 Results of parallel EAs for the HCSP. 66
6.9 Improvement factors and statistical analysis when using parallel CHC. . 66
6.10 pCHC and other methods for HCSP instances by Braun et al. (2001). . 67
6.11 Results of pμ-CHC for HCSP instances from Braun et al. (2001). 68
6.12 pμ-CHC improvement factors over pCHC. 69
6.13 pμ-CHC and other methods for HCSP instances by Braun et al. (2001). 70
6.14 pCHC results for new HCSP instances with dimension 1024×32. 71
6.15 pCHC results for new HCSP instances with dimension 2048×64. 71
6.16 pCHC results for new HCSP instances with dimension 4096×128. 72
6.17 pCHC results for new HCSP instances with dimension 8192×256. 72
6.18 pCHC improvements over traditional heuristics. 73
6.19 pμ-CHC results for new HCSP instances with dimension 1024×32. . . . 74
6.20 pμ-CHC results for new HCSP instances with dimension 2048×64. . . . 75
6.21 pμ-CHC results for new HCSP instances with dimension 4096×128. . . 75
6.22 pμ-CHC results for new HCSP instances with dimension 8192×256. . . 76
6.23 pμ-CHC improvements regarding the consistency classification. 77
6.24 pCHC and pμ-CHC comparative results for new HCSP instances. 78
6.25 pμ-CHC improvements with respect to pCHC. 79
6.26 pμ-CHC comparison with lower bounds (dimension 512×16). 82
6.27 Comparison between pμ-CHC results and lower bounds calculated for the preemptive HCSP (large instances). 84
6.28 Summary: gaps between pμ-CHC results and lower bounds for the preemptive HCSP version. 85

6.29 Trade-off between solution quality and execution time (pCHC and pμ-CHC). 87

7.1 Makespan results of pμ-CHC and Min-Min for the dynamic HCSP. . . . 97
7.2 Improvements of pμ-CHC using the rescheduling strategy over the traditional pμ-CHC and the two Min-Min variants. 98
7.3 Results of MOEAs for the multiobjective HCSP. 106
7.4 Efficacy metrics of MOEAs for the multiobjective HCSP. 106
7.5 Diversity metrics of MOEAs for the multiobjective HCSP. 107
7.6 Comparison of MOE-NSGA-II and pμ-CHC against previous evolutionary techniques for the multiobjective HCSP. 110

8.1 Summary of the main experimental results. 115

Acknowledgments

The research project reported in this thesis would not have been possible without the scientific, personal, and financial support of many people and organizations.

I would like to express my gratitude to my supervisors Héctor Cancela and Enrique Alba for guiding the research, and providing their helpful orientation, support and assistance in writing reports and publications. Very special thanks to all my partners at Centro de Cálculo and Instituto de Computación, Facultad de Ingeniería, Universidad de la República, specially my buddies at CeCal: Eduardo Fernández, Pablo Ezzatti, Martín Pedemonte, and Gerardo Ares; and IO: Franco Robledo, Alfredo Olivera, and Antonio Mauttone. Also thanks to all the good friends at Networking and Emerging Optimization, Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, España: Francisco Luna, Jamal Toutouh, José Manuel García Nieto, Juan José Durillo, Guillermo Molina, Gabriel Luque, and Francisco Chicano. Darío De León and Alexis Rodríguez worked in graduate projects related with the research reported in this thesis, and their work was useful for specific parts of the study.

I would like to thank all my family and friends who helped me get through all these years of Ph.D. studies, particularly when far away from home: mom and dad, Ale, my beloved Marcela, Auro and Aurora, and my good friends Chomba, Andrea, Raúl, and Paula. A special gratitude to Fefi, who assisted me with the thesis correction process.

I also acknowledge to several unknown reviewers of the related publications that provided helpful comments to improve the content of this thesis.

The research was partially supported by Comisión Sectorial de Investigación Científica (CSIC), Universidad de la República, Uruguay; Programa de Desarrollo de las Ciencias Básicas (PEDECIBA), Universidad de la República, Uruguay; and Agencia Nacional de Investigación e Innovación (ANII), Uruguay.

Chapter 1

Introduction

In the last fifteen years, the fast increase of the processing power of low-cost computers and the rapid development of high-speed networking technologies have boosted the use of distributed computing environments for solving complex problems. Starting from small clusters of homogeneous computers in the 1980's, as for today distributed computing environments include platforms formed by hundreds or thousands of heterogeneous computing resources widespread around the globe, providing the computing power needed for solving complex problems arising in many areas of application. Nowadays, a common platform for distributed computing usually comprises a heterogeneous collection of computers able to work cooperatively. At a higher level of abstraction, the expression *grid computing* has become popular to denote the set of distributed computing techniques that work over a large loosely-coupled virtual supercomputer, formed by combining together many heterogeneous components of different characteristics and computing power. This infrastructure has made it feasible to provide pervasive and cost-effective access to a collection of distributed computing resources for solving problems that demand large computing power (Foster and Kesselman, 1998).

A crucial problem when using such heterogeneous computing (HC) environments consists in finding a planning strategy or *scheduling* for a set of tasks to be executed. The goal of the scheduling problem is to optimally assign the computing resources by satisfying some efficiency criteria, usually related to the total execution time or resource utilization. Frequently, the scheduling strategy is devoted to optimize the *makespan*, a metric that quantifies the total execution time of the tasks in the system, even though alternative metrics have also been taken into account in many scheduling problem variants (e.g. *flowtime*, that evaluates the sum of the finishing times of the tasks, and several other metrics related to the resources *utilization*).

Scheduling problems on multiprocessor systems have been widely studied in operational research, and numerous methods have been proposed for finding accurate schedules in reasonable times (El-Rewini et al., 1994; Leung et al., 2004). In their classic formulation, scheduling problems assume a computing environment composed by homogeneous resources. However, in the 1990's decade the research community started to pay attention to scheduling problems on HC environments, specially due to the popularization of distributed computing and the growing use of *heterogeneous* clusters (Freund et al., 1994; Eshaghian, 1996). In the last ten years, significant effort has been made to study the scheduling problem on HC environments, since this platform provides the efficiency required for distributed and grid computing techniques.

Traditional scheduling problems are NP-hard (Garey and Johnson, 1979), thus classic exact methods are only useful for solving problem instances of reduced size. The research community has been searching for new scheduling techniques that are able to improve upon the traditional exact ones, whose low efficiency often makes them useless in practice for solving large-dimension scheduling problems in reasonable times. When dealing with large-dimension computing environments, ad-hoc heuristic and metaheuristic techniques have shown up as promising methods for solving the HC and grid scheduling problems. Although these methods do not guarantee success in computing an optimal solution for the problem, they get appropriate quasi-optimal schedules that usually satisfy the efficiency requirements for real-life scenarios, in reasonable times. Among a broad set of modern metaheuristic techniques for optimization, evolutionary algorithms (EAs) (Bäck et al., 1997) have emerged as flexible and robust methods for solving the *heterogeneous computing scheduling problem* (HCSP), achieving the high level of problem solving efficacy also shown in many other areas of application. Although they usually require longer execution times –in the order of few minutes– than ad-hoc heuristics, EAs consistently find better solutions than ad-hoc heuristic methods, so they are competitive schedulers for distributed HC and grid systems where large tasks –with execution times in the order of minutes, hours, and even days– are submitted for execution (Braun et al., 2001; Tupcouglu et al., 2002). In order to further improve the efficiency of EAs, parallel implementations became a popular option to speed up the search, allowing to reach high quality results in a reasonable execution time even for hard-to-solve optimization problems (Cantú-Paz, 2000; Alba, 2005).

EAs and other metaheuristics have been frequently applied to the HCSP and related problem variants in the last ten years. The most relevant proposals included Genetic Algorithms (GA) (Wang et al., 1997; Braun et al., 2001; Zomaya and Teh, 2001; Page and Naughton, 2005; Xhafa et al., 2008c), Memetic Algorithms (MA) (Xhafa, 2007), cellular MA (cMA) (Xhafa et al., 2008a), and also hybrid methods combining EAs with other optimization techniques. Two relevant works have obtained the best-known results when facing a set of low-sized de-facto standard HCSP instances using non-evolutionary metaheuristics: an hybrid combining Ant Colony Optimization (ACO) and Tabu Search (TS) (Ritchie and Levine, 2004) that took a long time –over 3.5 hours– to perform the search, and a hierarchic TS (Xhafa et al., 2008b) that used a time limit of 100 s. to run the scheduling algorithm. Other HCSP variants have been tackled using EAs, mostly remarkable the precedence-constrained task scheduling problem in multiprocessors (Wu et al., 2004; Boyer and Hura, 2005), real-time grid scheduling (Iordache et al., 2007), economy-based scheduling (Yu and Buyya, 2006), and other complex HCSP versions regarding many task attributes (Sugavanam et al., 2007; Braun et al., 2008).

Despite the numerous proposals on applying EAs and other metaheuristics to the HCSP and related variants, few works have tackled realistic instances in grid environments, mainly due to the inherent complexity of dealing with the underlying high-dimension optimization problem. In addition, few works studied parallel algorithms, in order to determine their ability to use the computing power available in large clusters or multicore computers to improve the search. Thus, there is still room to contribute in these lines of research by studying highly efficient parallel EA implementations that, by using simple operators, are able to scale-up and to deal with large-sized HCSP instances, eventually employing the available additional computational power of parallel and distributed computing environments.

This thesis presents the application of sequential and parallel EAs for solving the HCSP in distributed computing and grid environments. The work follows the line of research on studying evolutionary computation methods and parallel processing techniques for solving hard optimization problems, currently developed in Universidad de la República, Uruguay, and Universidad de Málaga, Spain. The research reported in this thesis was based in an initial conceptualization of parallel implementations of EAs, and the scheduling problem in distributed HC and grid environments. After performing a review of the state-of-the-art publications on the application of EAs and other metaheuristics to the HCSP and related problems, the conceptualization, design, and implementation of several EAs in their sequential and parallel variants was tackled. The experimental study was initially aimed to analyze the efficacy of the proposed EAs to solve a de-facto standard set of small-sized HCSP instances, by using simple search operators that allow the methods to scale up in order to face realistic large-sized problem instances. The analysis led to the proposal of a new parallel micro-CHC evolutionary algorithm (pμ-CHC) that uses specific operators in order to efficiently solve the HCSP. After that, the efficacy, efficiency, and scalability of the proposed EAs were evaluated for solving a new set of HCSP instances, far more complex than the ones existing in the present literature, designed by following a well-known methodology to model realistic HC environments such as large clusters and medium-size grid infrastructures. In addition, two lines of future work addressing variants of the HCSP have been initially studied: the scheduling problem in dynamic scenarios using the rescheduling strategy, and a multiobjective version of the HCSP devoted to simultaneously optimize the makespan and flowtime metrics.

The main contributions of this work are:

i. to analyze EA techniques for solving the scheduling problem on distributed heterogeneous environments,

ii. to apply parallel models of EAs that allow improving the quality of results and reducing the execution times,

iii. to provide new state-of-the-art methods for HC scheduling that can be used as building blocks for scaling to large distributed computing and grid systems,

iv. to introduce a new set of benchmark HCSP instances, specifically designed by following a well-known methodology, that model realistic HC environments such as large clusters and medium-size grid infrastructures, and

v. to introduce the pμ-CHC algorithm, a novel EA for solving the HCSP that combines a parallel subpopulation model with a focused evolutionary search using a micro population and a specific randomized local search method.

The experimental analysis shows that pμ-CHC is the new state-of-the-art algorithm for solving both standard problem instances and large unseen HCSP instances designed in this work. The new pμ-CHC algorithm outperforms previous results already reported in the related literature, and also shows a good scalability behavior when solving the new high dimension problem instances proposed in this work.

The content of the manuscript is structured as follows. Next chapter introduces the paradigm of evolutionary computation and presents the most important features of the EAs involved in this study. It also describes the application of parallel processing techniques in order to improve the efficiency and efficacy of EAs, and introduces the new pμ-CHC algorithm proposed in this work. The HCSP problem model and formulation are presented in Chapter 3, along with generic considerations about scheduling methods. The chapter also discusses the performance estimation methods and the HCSP instances already used in the related literature, just before introducing the new problem instances designed in this work in order to scale up to real-life-sized HC and grid environments. Finally, Chapter 3 also presents a brief description and review of several classic static heuristics and metaheuristics approaches using performance estimation, whose main ideas have been considered to design the EAs included in the study. Chapter 4 reviews and comments previous works that have proposed applying EAs to solve the HCSP and related variants in the last fifteen years. The review follows a chronological classification that allows identifying three stages regarding the scope of the proposals and the characteristics of the faced scenarios. Chapter 5 describes the implementation details of the serial and parallel EAs proposed in this study. It also presents MALLBA, the public C++ algorithmic environment on which the proposed EAs were implemented and the specific pμ-CHC algorithm applied to the HCSP. The experimental analysis and the discussion of obtained results are included in Chapter 6. It presents the parameter setting experiments, the numerical results and comparison with other techniques for standard and new large HCSP instances. The numerical analysis also includes the comparison with lower bounds computed for the preemptive case of the problem, and a study of the scalability and parallel performance behavior of the parallel EAs when solving the new large HCSP instances proposed in this work. The experimental results presented in Chapter 6 demonstrate the usefulness of the new pμ-CHC algorithm to efficiently compute accurate solutions for the HCSP. Chapter 7 presents the preliminary studies of two lines of research proposed as future work: the application of the pμ-CHC algorithm to solve two variants of the HCSP aimed at solving the scheduling problem in dynamic scenarios using the rescheduling strategy, and facing a multiobjective version of the HCSP devoted to optimize the makespan and flowtime metrics. Finally, the conclusions of the research are formulated in Chapter 8, along with a summary of the main contributions and the possible lines to continue the work in the near future.

Chapter 2

Parallel evolutionary algorithms

This chapter introduces the generic concepts about evolutionary computation techniques and describes the algorithms considered in this research for solving the HCSP. The chapter also presents the application of parallel processing techniques in order to improve the efficiency and efficacy of EAs, and introduces the new parallel micro-CHC evolutionary algorithm proposed in this work.

2.1 Introduction

Classic exact optimization methods –such as linear programming, branch and bound, dynamic programming, etc.– need to perform a superpolynomial number of operations when solving NP-hard optimization problems. Thus, those classic methods are generally only useful for solving problem instances of reduced size. In the last twenty-five years, the research community has been searching for new optimization techniques that are able to improve over the traditional exact ones, whose low efficiency often makes them useless in practice for solving large-dimension optimization problems in reasonable times. In this context, heuristics and metaheuristics techniques showed up as promising methods for solving NP-hard optimization problems (Glover and Kochenberger, 2003; Blum and Roli, 2003). Although heuristic and metaheuristics methods do not guarantee success in computing an optimal solution for the problem, they get appropriate quasi-optimal schedules that very often satisfy the efficiency requirements for real-life scenarios, in reasonable execution times. Among a broad set of modern metaheuristic methods for optimization, evolutionary computation techniques have emerged as flexible and robust methods for solving complex optimization problems in many areas of application like industry, mathematics, economy, telecommunications, and bioinformatics, among others (Goldberg, 1989a; Davis, 1991; Bäck et al., 1997).

Evolutionary computation techniques are stochastic methods that emulate the evolutionary process of natural species in order to solve optimization, search, learning, and other related problems. The idea of designing simulation methods for solving problems using the concepts of self-replication, self-modification, and evolution was suggested in the early 1960's by pioneers on information science and artificial intelligence such as Von Neumannn, Barricelli, Rechenberg, and others (Dyson, 1997). However, the fist algorithmic proposal of an evolutionary method dates from Holland, who suggested a *genetic* search procedure and presented the analytical background of evolutionary computation in the 1970's decade (Hayes-Roth, 1975).

As for today, the general term *evolutionary computation* includes a broad set of search and optimization techniques that use processes analogous to the natural evolution, but somehow different than the seminal concepts on self-replication and self-modification programs. In the last twenty-five years, EAs have been successfully applied for solving optimization problems underlying many real applications of high complexity. In order to considerably improve the efficiency of EAs, parallel implementations have been employed to enhance and speed up the evolutionary search. By splitting the computation in several processing elements, parallel evolutionary algorithms allow reaching high quality results in reasonable execution times even for hard-to-solve optimization problems (Cantú-Paz, 2000; Alba, 2005).

2.2 Evolutionary algorithms

The generic schema of an EA is shown in Algorithm 1. An EA is an iterative technique (each iteration is called a *generation*) that applies stochastic operators on a pool of individuals (the population P) in order to improve their *fitness*, a measure related to the objective function to optimize. Every individual in the population is the encoded version of a tentative solution of the problem. The initial population is either generated by a random method or by seeding the individuals using a specific heuristic for the problem. An evaluation function associates a fitness value to every individual, indicating its suitability to the problem. Iteratively, the probabilistic application of *variation operators* like the *recombination* of parts of two individuals and random changes (*mutations*) in their contents are guided by a selection-of-the-best technique to tentative solutions of higher quality. The stopping criteria usually involves a fixed number of generations or execution time, a quality threshold on the best fitness value, the detection of a stagnation situation, or a logical combination of all of the previously mentioned criteria. Specific policies are used to select the groups of individuals to recombine (the *selection* method) and to determine which new individuals are inserted in the population in each new generation (the *replacement* criterion). The EA returns the best solution ever found in the iterative process, taking into account the fitness function considered for the problem.

Algorithm 1 Schema of an evolutionary algorithm.

1: **initialize**($P(0)$)
2: `generation` $\leftarrow 0$
3: **while** not `stopcriteria` **do**
4: **evaluate**(P(`generation`))
5: `parents` $\leftarrow$ **selection**(P(`generation`))
6: `offspring` $\leftarrow$ **variation operators**(`parents`)
7: `newpop` $\leftarrow$ **replace**(`offspring`, P(`generation`))
8: `generation` ++
9: P(`generation`) $\leftarrow$ `newpop`
10: **end while**
11: **return** best solution ever found

In the last twenty-five years, EAs have been successfully applied for solving optimization problems underlying many real applications of high complexity. Next sections present specific instances and specializations of the generic algorithmic proposal described in Algorithm 1: genetic algorithm (GA), the CHC method, parallel EAs, and a novel parallel micro-CHC algorithm proposed in this work. All these methods have been applied in the research reported in this thesis for solving the HCSP.

2.3 Genetic algorithm

The classic formulation of a GA can be found in Goldberg (1989a) and Bäck et al. (1997). Based on the generic schema of an EA shown in Algorithm 1, the classic GA defines recombination and mutation as variation operators, applying them to the population of potential solutions in each generation. The recombination is used as the main operator to perform the search (exploiting the characteristics of suitable individuals), while the mutation is used as a (seldom-applied) secondary operator aimed at providing diversity for exploring different zones of the search space.

The simplest GA formulation –named *simple GA* (SGA)– was proposed by Goldberg (1989a). SGA uses a binary string for encoding solutions, the Single Point Crossover (SPX) as recombination operator and a mutation operator that randomly modifies selected positions in the solution encoding. Many other GA variants have been proposed in the literature by using alternative encodings, diverse recombination and mutation operators, or even different evolution models.

GAs are widely spread due to their versatility for solving combinatorial optimization problems. In this research, a traditional GA and other variants have been applied to solve the HCSP. The implementation details are presented in Chapter 5.

2.4 CHC algorithm

The CHC acronym stands for "Cross generational elitist selection, Heterogeneous recombination, and Cataclysmic mutation" (Eshelman, 1991). CHC is a specialization of a traditional GA that uses a conservative elitist selection strategy, that tends to perpetuate the best individuals in the population. CHC uses a special mating procedure: parents are randomly selected, but only those individuals which differ from each other by some number of bits are allowed to reproduce. The initial threshold for allowing mating is often set to 1/4 of the chromosome length. If no offspring is inserted into the new population during the mating procedure, this threshold is reduced by 1. The recombination operator in CHC is a special variant of the uniform crossover (UX), called half uniform crossover (HUX), which randomly swaps exactly half of the bits that differ between the two parent strings. CHC does not apply mutation, diversity is provided by applying a re-initialization procedure, using the best individual found so far as a template for creating a new population after convergence is detected.

Algorithm 2 presents a pseudo-code for the CHC algorithm, based on Eshelman's proposal, showing those features that make it different from traditional GAs: the highly elitist replacement strategy, the use of its own HUX recombination operator, the absence of mutation –which is substituted by a re-initialization operator– and the use of a mating restriction policy, that does not allow to recombine a pair of "too similar" individuals (considering a bit-to-bit distance function).

Algorithm 2 Schema of the CHC algorithm.

```
1: initialize(P(0))
2: generation ← 0;
3: distance ← (1/4) × chromosomeLength
4: while not stopcriteria do
5:   parents ← selection(P(generation))
6:   if distance(parents) ≥ distance then
7:     offspring ← HUX(parents)
8:     evaluate(offspring)
9:     newpop ← replacement(offspring, P(generation))
10:  end if
11:  if newpop == P(generation) then
12:    distance − −
13:  end if
14:  generation ++
15:  P(generation) ← newpop
16:  if distance == 0 then
17:    re-initialization(P(generation))
18:    distance ← (1/4) × chromosomeLength
19:  end if
20: end while
21: return best solution ever found
```

Although CHC is less known in the research community than other EA variants, it has been used with much success for solving difficult combinatorial optimization problems in our research groups. As examples, we can mention the design of robust network topologies (Nesmachnow et al., 2007), where several different metaheuristics –including CHC– were compared, and a multiobjective antenna placement problem (Nebro et al., 2007). In those previous works, CHC emerged as a very strong method for solving optimization problems, obtaining the best results among the studied algorithms.

The CHC method has been applied in this work for solving the HCSP. The implementation details are presented in Chapter 5.

2.5 Parallel evolutionary algorithms

The application of parallel processing techniques to EAs was suggested in early works in the 1960's decade, where the intrinsically parallel nature of the evolutionary process was identified as a key factor for the success of the evolutionary search (Holland, 1962; Bossert, 1967). However, no parallel implementations of EAs were proposed, mainly due to the lack of available parallel hardware in those years. Twenty years later, the first experiments with parallel implementations of genetic algorithms were reported during the 1980's decade (Grefenstette, 1981; Grosso, 1985). Since the massive popularization of parallel and high performance hardware architectures in the 1990's decade, many proposals of parallel evolutionary algorithms have been presented. Several reviews have been published about the topic (Nowostawski and Poli, 1999; Cantú-Paz, 2000; Alba et al., 2002), including a comprehensive review about the design and classification of parallel EAs by the author (Nesmachnow, 2002).

Parallel implementations became popular in the last fifteen years as an effort to improve the efficiency of EAs, while they also provide an improved search mechanism than often obtains better results than the sequential models. By splitting the population into several processing elements, parallel evolutionary algorithms (PEAs) allow reaching high quality results in a reasonable execution time even for hard-to-solve optimization problems (Cantú-Paz, 2000; Alba, 2005).

Three main models have been proposed in the related literature for designing PEAs, regarding the criterion used for the organization of the population: the master-slave model, the cellular model, and the distributed subpopulation model (Alba and Tomassini, 2002). These PEA models are briefly presented in the following subsections, before introducing the new parallel micro-CHC algorithm proposed in this work.

2.5.1 Master-slave PEAs

The master-slave model follows a classic functional decomposition of the EA, where different stages of the evolutionary process are performed in several computing resources (Alba, 2005). The evaluation of the fitness function is the main candidate to perform in parallel, since it usually requires larger computing time than the application of the evolutionary operators. Thus, a master-slave PEA is organized in a hierarchic structure: a master process performs the evolutionary search and controls a group of slave processes that evaluate the fitness function. Unlike the other two models for parallel EAs, the master-slave model has an identical algorithmic behavior than a sequential EA, since it works with a single panmictic population (i.e., no restrictions are established for the interactions between individuals). Figure 2.1 presents a graphical representation of the master-slave model for PEAs.

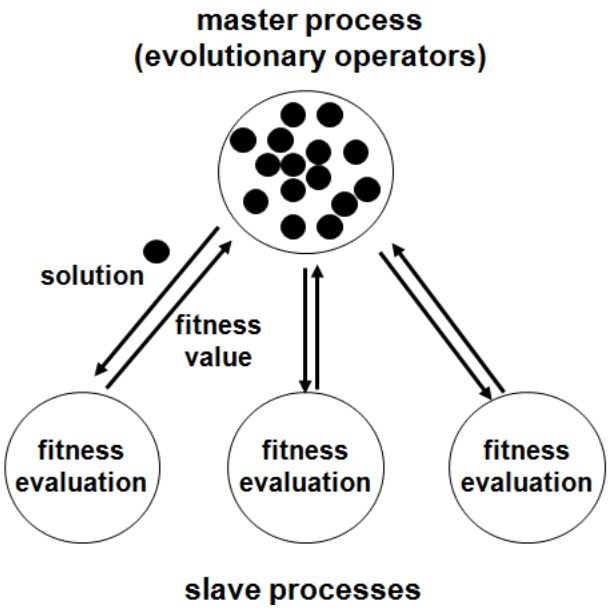


Figure 2.1: Diagram of a master slave PEA.

2.5.2 Cellular PEAs

The cellular model considers an underlying spatial structure for the individuals in the population (most usually a two-dimensional grid, while other connection topologies have been seldom used) (Alba and Dorronsoro, 2008). The interactions in the evolutionary search are restricted only to neighboring solutions. The propagation of good characteristics in the encoded solutions follows the *diffusion* model (Pettey, 1997), gradually spreading through the grid. These features provide the cellular model with an algorithmic behavior that differs from a traditional sequential EA. The limited interaction between individuals is useful for providing diversity in the population, often improving the efficacy of the evolutionary search. Figure 2.2 presents a graphical representation of the cellular model for PEAs, where I_{ij} represents the individual locates in the coordinates *i*-*j* in the two-dimensional grid.

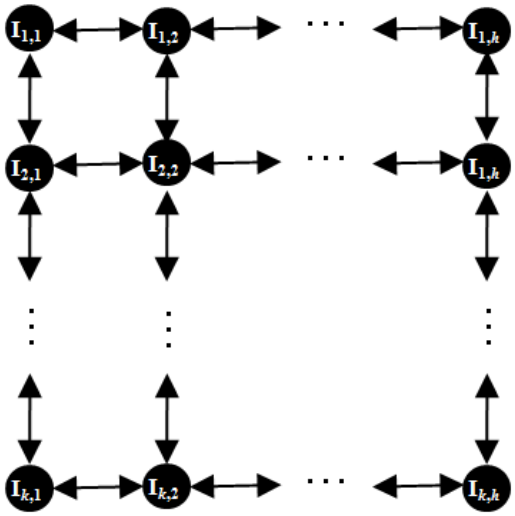


Figure 2.2: Diagram of a cellular PEA.

2.5.3 Distributed subpopulations PEAs

The distributed subpopulation model for PEAs (often also named *island model*) proposes to divide a panmictic population in several subpopulations (also called *islands* or *demes*), separated geographically from each other. Each deme runs a serial EA, so individuals are able to interact only with other individuals in the deme. Furthermore, an extra *migration* operator is defined: occasionally some selected individuals are exchanged among demes, introducing a new source of diversity in the evolutionary search. Figure 2.3 presents a graphical diagram of a distributed subpopulation PEA (Alba, 2005).

Algorithm 3 shows the generic schema for a distributed subpopulation PEA. Two conditions control the migration procedure: *sendmigrants* determines when the exchange of individuals takes place, and *recvmigrants* establishes whether a foreign set of individuals has to be received or not. These two conditions are separated in time in an *asynchronous* PEA, but they coincide in a *synchronous* model, when the send and receive operations are executed synchronically, one just after the other. *Migrants* denotes the set of individuals to exchange with some other deme, selected according to a given policy.

The generic schema for a distributed subpopulation PEA in Algorithm 3 explicitly distinguishes between *selection for reproduction* and *selection for migration*; they both return a selected group of individuals to perform the operation, but following potentially different policies. The *sendmigration* and *recvmigration* operators carry out the exchange of individuals among demes according to a connectivity graph defined over them, most usually an unidirectional ring.

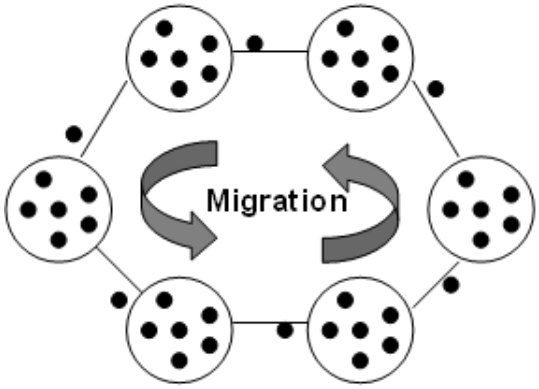


Figure 2.3: Diagram of a distributed subpopulation PEA.

Algorithm 3 Schema of a distributed subpopulation PEA.

```
1: initialize(P(0))
2: generation ← 0
3: while not stopcriteria do
4:    evaluate(P(generation))
5:    parents ← selection(P(generation))
6:    offspring ← reproduction operators(parents)
7:    newpop ← replace(offspring, P(generation))
8:    generation ++
9:    P(generation) ← newpop
      {Migration}
10:   if sendmigrants then
11:      migrants ← selection for migration(P(generation))
12:      sendmigration(migrants)
13:   end if
14:   if recvmigrants then
15:      inmigrants ← recvmigration()
16:      P(generation) ← insert(immigrants, P(generation))
17:   end if
18: end while
19: return best solution ever found
```

The PEAs proposed in this work for solving the HCSP follow the distributed subpopulation model. The implementation details are presented in Chapter 5.

2.6 Parallel micro-CHC algorithm

By splitting the global population, PEAs allow achieving high computational efficiency due to the limited interaction and the reduced population size within each deme. However, EAs quickly lose diversity in the solutions when using small populations, and the search suffers the well-known premature convergence effect, leading to a stagnation situation. The mating restriction policy and the re-initialization operator used in the CHC algorithm are usually not powerful enough to provide the required diversity to avoid premature convergence in the parallel model when using very small populations (i.e. less than 10 individuals per deme). Many alternatives have been proposed in the related literature to overcome the lose of diversity on EAs. In the quest for designing a fast and accurate version of the CHC algorithm for solving the HCSP, able to achieve high quality results in a reduced execution time, concepts from the micro-genetic algorithm (μ-GA) by Coello and Pulido (2001) were incorporated in this work in order to design a parallel micro-CHC algorithm.

Back in 2001, μ-GA was a novel proposal of EA in the context of multiobjective optimization, following previous works by Goldberg (1989b), Krishnakumar (1989) and Knowles and Corne (2000), aimed at speeding up the resolution of complex real-world problems. Years before, theoretical studies by Goldberg (1989b) hinted that an elitist evolutionary search is able to converge when using a reduced population size of only three individuals. Goldberg suggested a GA that uses a small population which evolves until reaching nominal convergence (i.e. when all individuals in the population are similar to each other). After that, a random reinitialization operator is applied in order to generate a new population, while keeping the best individuals from the previous cycle. The evolutionary search suggested by Golberg is rather similar to the CHC method later proposed by Eshelman, so devising a micro-CHC algorithm is a rather straightforward task.

The μ-GA by Coello and Pulido (2001) uses two populations to store memory along the search: the main population used in any EA, and a secondary population (also known as *elite population* or *external population*) which stores the best solutions found so far. The elite population allows keeping diversity at a low computational cost, by using the elite individuals to perform the population reinitialization after a certain low number of generations (in their proposal, Coello and Pulido (2001) applied the reinitialization after two to five generations pass when solving standard multiobjective problems).

The new parallel micro-CHC algorithm proposed in this work combines a distributed subpopulation parallel model of the original CHC structure by Eshelman (using HUX and mating restriction) with two key concepts from μ-GA: an external population that stores elite solutions found along the search, and an accelerated reinitialization using a specific randomized version of a well-known local search method to provide diversity within each subpopulation. The new algorithmic proposal uses an external population with three individuals, and a simple remove-of-the-worst strategy is used each time a new individual is inserted in the elite set. The accelerated reinitialization process applies both the CHC reinitialization operator and the local search method after a certain low number of generations (`MAX_COUNT_REINIT`, originally set to five generations) pass without inserting any offspring into the new population during the mating procedure. All these features have been proposed in this work to allow an efficient search when using a small population –i.e. less than 10 individuals– within each deme.

Algorithm 4 presents a pseudo-code for the new parallel micro-CHC algorithm (pμ-CHC), distinguishing those features that make it different from the traditional CHC algorithm: the use of a micro-population and an external elite population, the accelerated reinitialization, and the specific local search method employed to provide diversity.

Algorithm 4 Schema of the parallel micro-CHC algorithm.

```
1: initialize micro-population(P(0))
2: generation ← 0
3: counter ← 0 {counter for reinitialization}
4: distance ← initial_distance
5: elite_population = ∅
6: while not stopcriteria do
7:   parents ← selection(P(generation))
8:   if distance(parents) ≥ distance then
9:     offspring ← HUX(parents)
10:    evaluate(offspring)
11:    newpop ← replacement(offspring, P(generation))
12:  end if
13:  if newpop == P(generation) then
14:    distance − −
15:    counter ++
16:  end if
17:  generation ++
18:  P(generation) ← newpop
     {Reinitialization}
19:  if ((distance == 0) OR (counter == MAX_COUNT_REINIT)) then
20:    re-initialization(P(generation))
21:    local search(elite population)
22:    distance ← initial_distance
23:    counter ← 0;
24:  end if
25:  if best_solution_found then
26:    remove_worst(elite_population)
27:    insert(best_solution, elite_population)
28:  end if
29: end while
30: return best solution ever found
```

The pμ-CHC algorithm has been proposed in this work as an improved method for solving the HCSP, including some additional features in order to design an efficient and fully scalable implementation for tackling realistic large-dimension HCSP instances. The implementation details of the parallel micro-CHC for the HCSP are presented in Chapter 5.

2.7 Summary

This chapter has presented the main concepts about evolutionary computing techniques applied to solve optimization problems, and the specific evolutionary methods selected to be applied to the HCSP –GA and CHC– have been described. It also described the main ideas about the application of parallel processing techniques for improving the computational efficiency and the quality of results of EAs, briefly introducing the three parallel models of EAs that have been frequently used in the related literature. Finally, a direct contribution of this research has been introduced: the new parallel micro-CHC evolutionary algorithm, specially designed in order to provide a highly efficient search for complex optimization problems. pμ-CHC is inspired in previous works about GA using micro-populations and it incorporates specific features usually employed in the evolutionary multiobjective optimization field.

Chapter 3

Scheduling in heterogeneous computing environments

This chapter presents the HCSP mathematical formulation and concepts about the task execution time estimation in HC systems. It also discusses the HCSP instances already used in the related literature, as well as the need of new problem instances to scale up and model real-life HC and grid scenarios. Finally, it presents a brief description of some classic static heuristics and metaheuristic approaches using performance estimation, whose main ideas have been used in the design of the EAs included in this study.

3.1 Heterogeneous computing

A HC system is a well-orchestrated and coordinated set of processing elements, often called *resources*, *processors* or simply *machines*, interconnected by a network. The *heterogeneous* quality of the computing environment refers to the variable computational capabilities of the resources (e.g., CPU processing power, data transfer speed of the network, etc.). Usually, an HC system comprise a parallel and/or distributed suite of high performance machines, working together in order to provide support to cooperatively solve computing-intensive applications. Since the massive popularization of parallel and distributed computing in the 1990's decade, HC systems have emerged as a major paradigm for scientific and high performance applications that has been used to solve key problems with high computational requirements in many application areas (Khokhar et al., 1993; Freund et al., 1994; Eshaghian, 1996; Braun et al., 2000).

Nowadays, there exist mainly four types of HC systems: parallel systems in a single high performance supercomputer, distributed systems (usually connected in a Local Area Network), clusters of servers and workstations (often in a geographically distributed environment), and grid systems (widespread through the globe, with the potential scalability of using thousands of machines). Distributed clusters and grid systems have developed in the last decade as a powerful tool for tackling challenging hard-to-solve applications by using collaborative parallel-distributed computing strategies. While each of the previously described HC systems has its own characteristics, a crucial problem for all of them consists in finding a suitable tasks-to-machines assignment in order to satisfy some objectives related with efficiency, resource utilization, economic profit, and many other criteria. This problem, called *task assignment*, *task planning* or simply *scheduling*, has been an important focus in this line of research in the last twenty years.

Figure 3.1 shows an example of a distributed HC system composed of different (traditional and non-traditional) high performance computing platforms, and a centralized scheduler which performs the tasks-to-resources assignment. In some cases, a simplified version of the scheduling problem can be formulated by using information about the HC system (e.g., when all resources are managed by a single control point which accounts for full information about tasks and resource utilization). However, heterogeneity usually poses a hard challenge for the scheduling application, specially in fully distributed HC systems, where few information about tasks and resources is available, and different resource utilization policies may be locally applied.

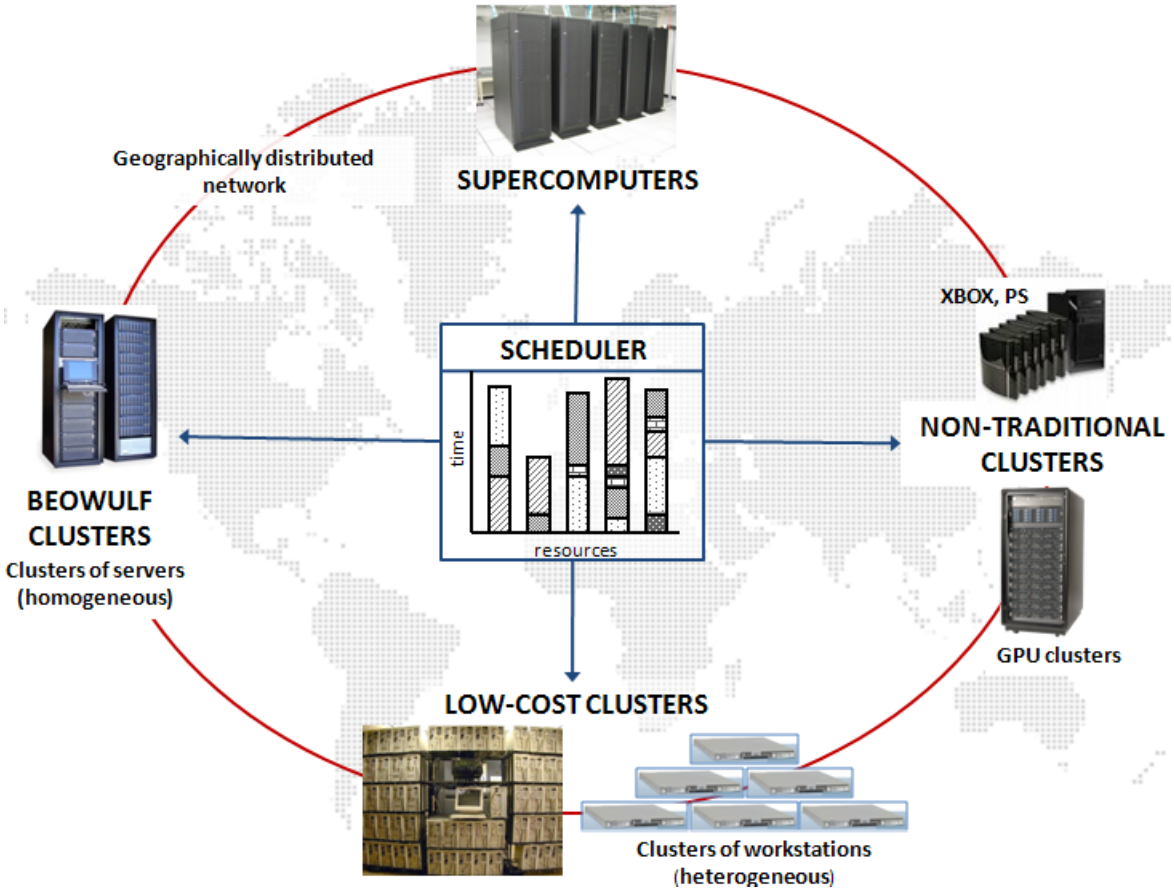


Figure 3.1: Scheduling in an HC environment.

3.2 HCSP formulation

Let an HC system be composed of many computers, also called *processors* or simply *machines*, and a set of tasks with variable computational requirements, to be executed on the system. A task is the atomic unit of workload to be assigned to a computing resource, so a task cannot be divided in smaller chunks, nor interrupted after it is assigned (the scheduling problem follows a *non-preemptive* model). The execution times of any individual task vary from one machine to another, so there will be a competition among tasks for using those machines able to execute each task in the shortest time.

The general formulation of a scheduling problem proposes to find a tasks-to-machines assignment in order to satisfy some objectives. Given that using the computational resources does not come for free, most scheduling problems mainly concern about time, trying to minimize the total time spent to execute all tasks. Several properties or attributes can be assigned to each task (i.e. CPU and memory requirements, priority, execution time deadline, etc.), but the simplest model used for scheduling, and also the most common one, considers only the execution time of each task in each computing resource as the most relevant property. The most usual metric to minimize in this model is the *makespan*, defined as the time spent from the moment when the first task starts its execution to the moment when the last task is completed. However, many other performance metrics have been considered in scheduling problems, such as the *economic cost* of executing an application and the *quality of service*, which is specially pertinent in grid infrastructures (Leung et al., 2004). Other relevant metrics, from the performance optimization point of view, include the *resource utilization*, the *throughput* and also the *economic profit* of a certain infrastructure. Beyond all those single-objective approaches, the scheduling problem is an obvious multiobjective problem when considering several task properties or several performance metrics in conflict with each other.

This work mainly concerns the optimization of the makespan metric, and the following formulation presents the mathematical model for the HCSP aimed at minimizing the makespan:

- Given an HC system composed of a set of computational resources (machines) $P = \{m_1, m_2, \ldots, m_M\}$ (dimension M), and a collection of tasks $T = \{t_1, t_2, \ldots, t_N\}$ (dimension N) to be executed on the HC system,
- let there be an *execution time function* $ET : P \times T \rightarrow \mathbf{R}^+$, where $ET(t_i, m_j)$ is the time required to execute the task t_i in the machine m_j,
- the goal of the HCSP is to find an assignment of tasks to machines (a function $f : T^N \rightarrow P^M$) which minimizes the *makespan*, defined in Equation 3.1.

$$\max_{m_j \in P} \sum_{\substack{t_i \in T: \\ f(t_i)=m_j}} ET(t_i, m_j) \tag{3.1}$$

The previous model does not account for dependencies among tasks: the formulation assumes that all tasks can be independently executed, disregarding the execution order. Even though it is a simplified version of the more general scheduling problem that accounts for task dependencies, the independent task model is specially important in distributed environments such as supercomputing centers and grid infrastructures. Independent-task applications frequently appears in many lines of scientific research, and they are specially relevant in Single-Program Multiple-Data (SPMD) applications used for multimedia processing, data mining, parallel domain decomposition of numerical models for physical phenomena, etc. In addition, the independent tasks model also arises when different users submit their (obviously independent) tasks to execute in a grid computing service -such as Berkeley's BOINC, Xgrid, TeraGrid, EGEE, etc. (Berman et al., 2003)-, and in *parameter sweep applications*, structured as a set of multiple experiments, each one executed with a different set of parameter values. Thus, the relevance of the HCSP version faced in this work is justified due to its significance in realistic distributed HC and grid environments.

3.2.1 Problem model considerations

Scheduling policies and algorithms are classified in *static*, *dynamic* or *adaptive*, regarding the moment in which the scheduling decisions are taken. In a static method, the scheduler gathers all the available information about tasks and resources *before* the execution: the resource availability and the collection of tasks is completely known a-priori, and it is assumed that no task arrives after the planning is done. The scheduler takes early decisions, and the task-to-resource assignment is not allowed to change during the execution. Static schedulers require an accurate estimation of the execution time for each task on each machine, which is usually achieved by performing task profiling and statistical analysis of both submitted workloads and resource utilization (see Section 3.3). Although static scheduling does not account for infrastructure failures or unexpected overload situations, auxiliary mechanisms such as rescheduling can be used to mitigate the effect of disruptive events. By contrast, dynamic scheduling methods allow planning when tasks are admitted to arrive dynamically, providing an efficient way to deal with problematic events such as unexpected overload and resource crashes. Dynamic schedulers try to overcome the difficulties of execution time estimation by taking the scheduling decisions while the application is running. They rely on information about the current state of the HC system to properly take the adequate decisions about where to dispatch an incoming task, or where to reallocate it when a failure occurs, in order to maintain a good load balancing pattern. In a higher level of complexity, adaptive scheduling techniques apply the dynamic assignment model, but they also incorporate prediction techniques trying to take the most adequate scheduling decisions after foreseeing the system behavior. Adaptive schedulers often require complex procedures to know in advance the metrics needed to perform adaptation, such as resource load and availability. They also frequently need to forecast the tasks execution times in order to increase their planning efficiency. Nevertheless, when using forecasting techniques, the resulting tool is accurate, but it is also often application-specific, with restricted use in general-purpose domains.

This work proposes using parallel EAs to solve the *static* HCSP. Static scheduling has it own areas of specific application, such as planning in distributed clusters and HC multiprocessors, and also analyzing the resource utilization for a given hardware infrastructure. Static scheduling also provides a first step for solving more complex scheduling problems arising in distributed and dynamic environments: static results can be used as a reference baseline to determine if a dynamic scheduler is taking the right decisions about using the available resources in the system. In addition, an efficient static planner that obtains accurate results for high dimension scheduling problems, can be the building block to develop a powerful dynamic scheduling application, able to deal with the increasing complexity of nowadays grid infrastructures.

3.2.2 HCSP computational complexity

The heterogeneous computing scheduling problem is NP-hard. HCSP is a generalization of the Minimum Resource Constrained Scheduling Problem (problem SS8 in the classification by Garey and Johnson (1979)) where no resource constraints are formulated, and the *task length* ($l(t_i)$ in SS8) is replaced by the execution time function $ET(t_i, m_j)$. The HCSP has been also named as *Minimum Multiprocessor Scheduling Problem* in the compendium of NP optimization problems by Crescenzi and Kann (1999).

The dimension of the HCSP search space is M^N for a problem scenario with N tasks and M machines. This fact means that the number of possible schedules increases exponentially with the number of tasks. Even for (unrealistic) very small problem instances, the dimension of the HCSP search space is really huge (more than a million schedules for $M = 4$, $N = 10$, and almost 100 billion schedules for $M = 5$, $N = 20$). The high computational complexity of HCSP implies that classical exact methods, such as branch and bound, dynamic programming or linear programming, are only useful for solving problem instances of reduced size. In this context, ad-hoc heuristic and metaheuristic techniques are promising methods for solving the HCSP and related problems in distributed HC and grid environments, since they are able to get accurate suboptimal schedules that satisfy the efficiency requirements for real-life scenarios, in reasonable times.

3.3 Execution time estimation

It is quite impossible to exactly evaluate the execution time of a computer program on a specific computing resource. Even though the computational complexity of a program may be known, it only provides a bound function (depending on the input data size), which is independent of the underlying hardware. In addition, the computational power (in CPU cycles) of any processor depends on many structural components, so an exact value cannot be achieved without considering several design issues. Approximation methods are needed in order to estimate the execution time of any given program on a specific computing resource.

Execution time estimation is a common technique applied to model the execution time of tasks on a computer that has been used in HC static scheduling since the early 1990s (Yang et al., 1993; Singh and Youssef, 1996). This technique relies on estimation methods such as task profiling (used to predict the computational requirements of a given task), benchmarking (used to estimate the computing power of a machine) and statistical analysis of both submitted workloads and resource utilization, in order to provide an accurate prediction of the execution time of a given task in a specific machine. The estimation scenario is a feasible and realistic one, with only a few assumptions, given that the computational capacity of machines are quite easy to estimate, and the computational requirements of each task can be approximated by exploiting specifications provided by the users, analyzing historic data, and/or using more advanced prediction models. Researchers have stated that predicting the task execution times is useful to guide the resource selection performed by a scheduling method and also to achieve load balancing on HC environments (Li et al., 2004).

This section introduces the *expected time to compute* performance estimation model. It also discusses two methods and parameters used for generating matrices that represent the execution time function for a set of tasks in an HC system in the estimation model.

3.3.1 Expected time to compute estimation model

In 2000, Ali et al. (2000) presented the *expected time to compute* (ETC) performance estimation model, which has been widely used in the research about HC scheduling in the last nine years. ETC provides an estimation for the execution time of a collection of tasks in an HC system, taking into account three key properties: machine heterogeneity, task heterogeneity and consistence.

Machine heterogeneity evaluates the variation of execution times for a given task across the HC resources. An environment comprised by similar computing resources will be represented by low machine heterogeneity, while high machine heterogeneity represents generic HC systems, integrated by computing resources of different type and power. *Task heterogeneity* represents the degree of variation among the execution times of tasks for a given machine. High task heterogeneity describes those scenarios in which different types of applications are submitted to execute in the HC system, from simple programs to large and complex tasks which require large CPU times to be performed. On the other hand, when the complexity, and thus the computational requirements of the tasks are quite similar, they shall have rather similar execution times for a given machine. This situation is modeled by a low task heterogeneity scenario.

The ETC model also considers a second classification, trying to reflect the characteristics of realistic scenarios. In a *consistent* ETC scenario, whenever a given machine m_j executes any task t_i faster than other machine m_k, then machine m_j executes all tasks faster than machine m_k. This situation corresponds to an ideal case where the execution time of each task is mainly determined for the computational power of each machine, since tasks are supposed to perform at the same rate in all machines. A scenario with this structure seems to be unrealistic for general purpose applications, but it captures the reality of many SPMD applications executing with local input data. An *inconsistent* ETC scenario lacks of structure among the computing demands of tasks and the computing power of machines, so a given machine m_j may be faster than another machine m_k when executing some tasks, and slower for others (i.e. execution time values are uncorrelated). This category represents the most generic scenario for a distributed HC infrastructure that receives many kinds of tasks, from easy-to-solve programs to very complex parallel models. In addition, a third category of *semi-consistent* ETC scenarios is included, to model those inconsistent systems that include a consistent subsystem. In this last category, even though there is not a predefined structure on the whole sets of tasks and machines in the HC system, some of them behave like a consistent HC system.

Table 3.1 presents the twelve possible combinations of heterogeneity types and consistency classifications in the ETC model by Ali et al. (2000) (`hi` stands for high heterogeneity, `lo` for low heterogeneity, and the consistency categories are named for the correspondent initial letter).

heterogeneity		**consistency**		
task	machine	consistent	inconsistent	semiconsistent
`hi`	`hi`	`c/hihi`	`i/hihi`	`s/hihi`
`hi`	`lo`	`c/hilo`	`i/hilo`	`s/hilo`
`lo`	`hi`	`c/lohi`	`i/lohi`	`s/lohi`
`lo`	`lo`	`c/lolo`	`i/lolo`	`s/lolo`

Table 3.1: Heterogeneity and consistency combinations in the ETC model.

In order to effectively model real-life HC systems, the ETC prediction does not only account for the explicit running time of a task in a certain machine. Other relevant factors in parallel and distributed computing are also included in the performance estimation, such as the time needed to move the executable files to the target machine, the time needed to gather and transfer the input data for the computation, and eventual short communications. All these overheads are supposed to be included in the ETC estimation.

3.3.2 Methods for generating ETC scenarios

In their pioneering work, Ali et al. (2000) proposed two methods for designing random ETC matrices to represent diverse HCSP scenarios: the *range based* method and the *coefficient of variation* method, which only differ in the mathematical model used to define the heterogeneity values for tasks and machines.

The range based method defines two ranges: $(1, R_{mach})$ for machine heterogeneity, and $(1, R_{task})$ for task heterogeneity. Heterogeneity values for machines (τ_M) and tasks (τ_T) are randomly sampled using an uniform distribution, and the ETC for task i in machine j is computed by $ETC(i,j) = \tau_T(i) \times \tau_M(j)$. The authors suggested using the parameter values presented in the first row of Table 3.2 –selected to model relevant scenarios for the Management System for Heterogeneous Networks project (Hensgen et al., 1999)– to generate HCSP scenarios. However, no specific HCSP test suites were generated by Ali et al. (2000).

The range based method was later used to create a test suite of random HCSP scenarios by Braun et al. (2001), using the parameter values showed in the second row of Table 3.2 (the details about these instances are presented in Section 3.4.1).

model	**task heterogeneity**		**machine heterogeneity**	
	low	**high**	**low**	**high**
Ali et al. (2000)	$R_{task} = 10$	$R_{task} = 100000$	$R_{mach} = 10$	$R_{mach} = 1000$
Braun et al. (2001)	$R_{task} = 100$	$R_{task} = 3000$	$R_{mach} = 10$	$R_{mach} = 1000$

Table 3.2: Parameters of ETC models.

The *coefficient of variation* method uses the quotient between the standard deviation and the mean of execution times values as a measure of machine and task heterogeneity, and the ETC values are randomly generated using the uniform distribution.

The HCSP instances used to evaluate the parallel EAs proposed in this work follow the ETC model by Ali et al. Their characteristics are commented in the next section.

3.4 HCSP instances

Although the research community has faced the HCSP in the past, there do not exist standard benchmarks or test suites for the problem (Theys et al., 2001; Dong and Akl, 2006). A test suite of twelve randomly generated HCSP instances presented by Braun et al. (2001) have been employed to evaluate heuristic and metaheuristic methods for the HCSP in a large number of publications. Those instances have been also adopted as a reference baseline for designing a complete set of large HCSP instances in this work, so they are commented in section 3.4.1. Some other authors, like Page and Naughton (2005) generated their own test suite of random instances to evaluate the effectiveness of heuristic algorithms for the HCSP, while arguing that "it is not clear what characteristics a typical task would exhibit". Several other works used randomly generated HCSP instances, but researchers did not often put much effort in describing the methodology for creating the random scenarios used. It is also worth noting that none of the previous proposals have scaled up the dimension of the designed problem instances in order to model realistic grid environments, with the exception of two works by Xhafa et al. (2007b) and Boyer and Hura (2005).

Xhafa et al. (2007b) generated four inconsistent ETC matrices with 4096 tasks and 256 machines with high task and machine heterogeneity, arguing that "these are usually the most difficult instances to solve" to evaluate several EAs. Boyer and Hura (2005) used scenarios up to 1000 machines and 5000 tasks for evaluating an iterated randomized search method for the HCSP. However, neither Xhafa et al. (2007b) nor Boyer and Hura (2005) published the HCSP instances used.

Next subsections describe the HCSP instances already used in the related literature, and present some details about nowadays grid platforms, before introducing the new set of problem instances specifically designed in this work to challenge state-of-the-art scheduling techniques.

3.4.1 Instances from Braun et al. (2001)

Braun et al. (2001) presented an HCSP test suite with twelve instances generated using the range based method. All the instances have 512 tasks and 16 machines, and they combine the three ETC model properties (task and machine heterogeneity, and consistency). Instead of using the previously proposed ETC parametrization by Ali et al. (2000), the authors suggested the upper bounds for machine and task heterogeneity intervals presented in the second row of Table 3.2. These values were selected to model several characteristics of the prediction methods used in Armstrong et al. (1998), but the authors pointed out that their election was quite arbitrary, and suggested that researchers may substitute in their own values to generate instances that model other specific situations of interest. However, the commented test suite has become a de-facto standard benchmark to evaluate algorithms for solving the HCSP.

The instances from Braun et al. (2001) are labeled with a name with the pattern `d_c_MHTH.0`, where `d` indicates the distribution function used to generate the ETC values (`u`, for the uniform distribution), and `c` indicates the consistency type (`c` for consistent, `i` for inconsistent, and `s` for semiconsistent). `MH` and `TH` indicate the heterogeneity level for tasks and machines respectively (`lo` for low heterogeneity, and `hi` for high heterogeneity). The final number after the dot (`0`) refers to the number of test cases (initially, several suites were generated, but only the class `0` gained popularity).

3.4.2 Grid infrastructures

In the last decade, distributed computing environments have grown at a fast pace. The previously described HCSP instances from Braun et al. (2001) were conceived for modeling multiprocessor HC systems, and so they do not capture the reality of nowadays large-scale cluster computing and grid infrastructures. As an example, the Berkeley Network of Workstations project surpassed the one-hundred-processors milestone in the middle of the 1990's decade (Anderson et al., 1995). Nowadays, modern grid initiatives use platforms with more than 1,000 processors, while hierarchical computing grids distributed worldwide and volunteer-based distributed computing platforms manage more than 100,000 computing resources (Berman et al., 2003). Table 3.3 presents a brief description of sampled grid and volunteer-based distributed computing infrastructures, showing the large number of processing resources usually involved in those platforms.

name	location	processors	comment/URL
medium-sized grids			
EELA	Europe-LA	~ 750	regional grid, `www.eu-eela.eu`
Grid5000	France	$> 3,000$	national grid, `www.grid5000.fr`
large grids			
OSG	USA	$> 30,000$	national grid, `www.opensciencegrid.org`
TeraGrid	USA	$> 40,000$	national grid, `www.teragrid.org`
EGEE	Europe	$> 80,000$	continental grid, `www.eu-egee.org`
WLCG	Europe	$> 100,000$	CERN worldwide grid, `www.cern.ch/lcg`
volunteer distributed computing platforms			
SETI@home	worldwide	$> 350,000$	`setiathome.ssl.berkeley.edu`
BOINC	worldwide	$> 500,000$	`boinc.berkeley.edu`
Folding@Home	worldwide	$> 250,000$	`folding.stanford.edu`

Table 3.3: Details of sampled grid and volunteer distributed computing platforms.

EELA-2 (eScience grid facility for Europe and Latin America) and Grid500 are examples of medium-sized grids. EELA-2 is currently the largest grid initiative involving Latin America, with less than a thousand processors (Brasilero et al., 2008). Grid5000 has 1597 nodes (8 families, 17 systems) with a total number of 3,000 processors, but the experimental scenarios for scheduling algorithms usually consider less than 250 processors, often grouped in few heterogeneous classes (Caniou and Gay, 2008; Mohamed and Epema, 2008). However, some large scale experiments involving more than 1000 processors have been planned to analyze middleware resource managers and batch schedulers (Grid5000, 2009). On the other hand, large grid infrastructures usually have a hierarchical structure, such as TerGrid and WLCG, the largest computing grid in the world in 2009, with almost 100,000 projected processors at CERN, Switzerland, and more than 100,000 additional processors distributed worldwide (WLCG, 2009). In these large-scale organizations, heterogeneity is only handled on high-level schedulers, while local schedulers perform the intra-site task allocation on homogeneous machines. Volunteer-based distributed computing platforms usually comprehend a huge number of computing resources, but the scheduling is often performed using simple methods for tasks-to-processors assignment. SETI@home was a pioneering initiative of volunteer distributed computing launched in the last years of the 20th century. Today, it involves more than 350,000 active computers, while the number of total resources used since the start of the project is near to 2 million (SETI@home, 2009). BOINC is nowadays the largest volunteer distributed computing platform, with more than half a million active computers (BOINC, 2009), and today provides the support for the SETI@home project. Folding@home is the first computing project ever to cross the 4 petaFLOPS milestone (reaching 7 petaFLOPS in August 2009), by using the cooperative effort of more than 250,000 PS3 processors and GPU units (Folding@home, 2009).

The previous description of modern grid infrastructures shows that new problem instances, larger than the ones proposed by Braun et al. (2001), are needed in order to make cutting-edge research on the scalability of scheduling algorithms for solving real-life scenarios.

3.4.3 New HCSP instances

Apart from the proposals previously commented in Section 3.4.1, there has been little effort to define a standard test suite for HC scheduling. Even today, when grid scheduling has been the focus of many works, researchers have been using the test suite from Braun et al. (2001) or proprietary instances, often generated without following a methodological basis. One of the main objectives of the work reported in this thesis consists in studying the scalability of new methods to solve the HCSP (i.e. how the solution quality achieved using a fixed execution time varies when the instances dimension grows). In order to perform the analysis, this work introduces a test suite of large HCSP instances designed following the methodology for execution time estimation proposed by Ali et al. (2000).

The new HCSP instances were created using a random generator program, implemented in the C language using the standard C libraries `stdlib.h` and `math.h`, without requiring any additional software. The generator implements the range based method from Ali et al. (2000), regarding the relevant scenario parameters: dimension (number of tasks and machines), task and machine heterogeneity, consistency, and two different parametrization models of ETC.

The input parameters for the HCSP instances generator program and its execution syntaxis are presented in Figure 3.2.

```
Syntaxis: ./generator <num_tasks> <num_machines> <task_het>
        <machine_het> <consistency> [model] [type] [seed]
Required parameters:
<num_tasks>: number of tasks (integer).
<num_machines>: number of machines (integer).
<task_het>: task heterogeneity level (0-Low, 1-High).
<machine_het>: machine heterogeneity level (0-Low, 1-High).
<consistency>: consistency type (0-Consistent,
             1-Partially consistent, 2-Inconsistent).
Optional parameters:
[model]: heterogeneity model (0-Ali et al., 1-Braun et al.).
        Braun et al. model assumed as the default option.
        Heterogeneity ranks (tasks, machines):
           Ali et al. (10-100000,10-100),
           Braun et al. (100-3000,10-1000).
[type]: task execution time type (0-real, 1-integer).
        Real type assumed as default.
[seed]: seed for the random number generator (integer).
        Allows replicating the generation of instances.
```

Figure 3.2: Details of the HCSP instances generator.

The output file format is similar to the one employed by Braun et al. (2001): a column vector of $N \times M$ floating point numbers (or integers) that represents the ETC matrix, ordered by task identifier. Figure 3.3 presents an example of the format for an HCSP scenario with `tT` tasks and `mM` machines.

```
ETC(t1,m1)
ETC(t1,m2)
...
ETC(t1,mM)
ETC(t2,m1)
ETC(t2,m2)
...
ETC(tT,m1)
...
ETC(tT,mM)
```

Figure 3.3: Format of the HCSP instance files.

The HCSP test suite generated in this work includes instances with diverse complexity. The *small-sized* instances include twelve instances with 512 tasks and 16 machines generated using the ETC parameters from Ali et al. (2000), and also problem instances up to 1,024 tasks and 32 processors. The *medium-sized* instances include up to 4,096 tasks and 128 machines, and they are considered as representative of large multiprocessors, medium-size clusters of computers, and small grid systems. The group of *large-sized* instances includes scenarios with up to 8,192 tasks and 256 processors, a dimension that represents large clusters and medium-size grid systems. For each dimension (except 512×16), twenty-four HCSP instances were generated regarding all the heterogeneity and consistency combinations, twelve of them using the parameter values from Ali et al. (2000), and twelve using the values from Braun et al. (2001); in order to avoid biased results in the experimental analysis. The instances are named following the previously presented convention for the instances by Braun et al. (2001): the names have the pattern `M.d_c_MHTH`, where the first letter (`M`) describes the heterogeneity model used (`A` for Ali, and `B` for Braun). The number `0` originally included in the last position of the name for the instances by Braun et al. (2001) was omitted in the new HCSP instances, since only one suite was generated for each problem dimension and ETC parametrization.

Since the new test suite was designed following a well-known methodology, the problem instances maintain the relevant properties of the ETC performance estimation model by Ali et al. (2000). By including a more comprehensive set of scenarios, the test suite allows performing studies aimed at obtaining a better characterization of new scheduling methods, specially to solve large HCSP instances that better model present distributed HC and grid systems. The new set of large-sized HCSP instances poses a real challenge to scheduling methods: the size of the search space increases from 3.2×10^{616} possible schedules in the set of instances with dimension 512×16 by Braun et al. (2001) to 2×10^{19728} for the largest new HCSP instances (dimension 8192×256). Thus, the new set of HCSP instances is useful to analyze the efficacy of scheduling methods when the problem instances grow.

The problem instances and the generator code are presented in Appendix A. The HCSP instances and the generator program are also publicly available to download at the HCSP website `http://www.fing.edu.uy/inco/grupos/cecal/hpc/HCSP`.

3.5 Static scheduling using execution time estimation

The relevance of the scheduling problem on HC environments led to the proposal of many deterministic and randomized methods for solving the HCSP. A plethora of works presented deterministic heuristics through the 1990s, but the most complete review was the comprehensive study by Kwok and Ahmad (1999), who extensively classified, described and compared twenty-seven classic scheduling algorithms for the static task allocation problem. This subsection briefly describes the most popular deterministic methods and metaheuristic approaches using execution time estimation, whose main ideas have been used for designing the EAs for the HCSP proposed in this work.

3.5.1 Traditional scheduling heuristics

The class of *list scheduling techniques* comprises a large set of deterministic static scheduling methods that work by assigning priorities to tasks based on a particular ad-hoc heuristic. After that, the list of tasks is sorted in decreasing priority and each task is assigned to a processor, regarding the task priority and the processor availability. Algorithm 5 presents the general schema of a list scheduling method.

Algorithm 5 Schema of a list scheduling algorithm.

```
1: while tasks left to assign do
2:    determine the most suitable task according to the chosen criterion
3:    for each task to assign, each machine do
4:       evaluate criterion (task, machine)
5:    end for
6:    assign the selected task to the selected machine
7: end while
8: return task assignment
```

Since the pioneering work by Ibarra and Kim (1977), where the first algorithms following the generic schema presented in Algorithm 5 were introduced, many list scheduling techniques have been proposed in order to provide easy methods for tasks-to-processors scheduling. This class of methods has also often been employed in hybrid algorithms, most usually combined with EAs, with the objective of improving the search of metaheuristic approaches for the HCSP and related scheduling problems.

The simplest category of list scheduling heuristics applied to minimize the makespan include:

- **Shortest Job to Fastest Resource** (SJFR) is a simple technique that sorts the tasks by increasing ETC and assigns them to the available resources, decreasingly sorted by their computing capacity. The method is devoted to reduce the average response time of tasks.

- **Longest Job to Fastest Resource** (LJFR) is motivated by results in bin-packing problems, indicating that a simple first-fit algorithm achieves better packing when sorted in decreasing size. Thus, LJFR assigns the longest tasks, decreasingly sorted by ETC, to the available resources, decreasingly sorted by their computing capacity.

- **Opportunistic Load Balancing** (OLB) considers the set of tasks sorted in an arbitrary order, and assigns them to the next machine that is expected to be available, regardless of the ETC for each task on that machine. OLB is a simple method that intends to maximize the resource utilization, keeping all the machines as busy as possible. However, as it does not account for the expected tasks execution times, the resulting schedules will often have high makespan values.

- **Minimum Execution Time** (MET) considers the set of tasks sorted in an arbitrary order, and assigns them to the machine with lower ETC for that task, regardless of the machine availability. MET assign each task to its best machine, but since it does not account for the resource availability, severe load imbalance should arise, specially when applied in consistent ETC instances.

- **Minimum Completion Time** (MCT) tries to combine the benefits of OLB and MET. Considering the set of tasks sorted in an arbitrary order, MCT assigns each task to the machine with the minimum ETC for that task. By assigning tasks to machines that do not have the minimum ETC for them, MCT intends to avoid those cases in which both OLB and MET find poor schedules.

In addition, several methods have proposed trying to overcome the inefficacy of simple static heuristics by taking into account more complex and holistic criteria to perform the task mapping, and then reduce the makespan values. Some of the most popular heuristics include:

- **Min-Min** greedily picks the task that can be completed the soonest. The method starts with a set U of all *unmapped* tasks, calculates the MCT for each task in U for each machine, and assigns the task with the minimum overall MCT to the best machine. The mapped task is removed from U, and the process is repeated until all tasks are mapped. Min-Min improves upon the MCT heuristic, since it does not consider a single task at a time but all the unmapped tasks sorted by MCT, and the availability status of the machines is updated by the least possible amount of time for every assignment. This procedure leads to more balanced schedules and generally also allows finding smaller makespan values than other heuristics, since more tasks are expected to be assigned to the machines that can complete them the earliest.

- **Max-Min** follows an iterative procedure similar to Min-Min, but assigns the task with the overall *maximum* MCT to the best machine. The method tries to minimize the penalties from performing tasks with longer ETC, attempting to exploit the capability of executing the shorter tasks in parallel. In scenarios with high task heterogeneity, Max-Min could lead to find more load balanced schedules than Min-Min, and also better makespan values.

- **Duplex** is a combination of the Min-Min and Max-Min heuristics, which performs the previously described techniques and uses the better solution. Duplex has been applied to overcome the problems of the two previously described methods, by exploiting the conditions in which either Min-Min or Max-Min are able to find accurate schedules.

- **Sufferage** identifies in each iteration step the task that if it is not assigned to a certain host, it will *suffer* the most. The *sufferage value* is computed as the difference between the best MCT of the task and its second-best MCT. Sufferage gives precedence to those tasks with high sufferage value, assigning them to the machines that can complete them at the earliest time.

- **XSufferage** was proposed to fix a specific problem of Sufferage that arises when the best and second best MCTs are nearly identical for a given task. In this situation –rather frequent when using clusters with near-identical performance resources–, the sufferage value is close to zero, and Sufferage will defer the task, disregarding execution time considerations. To fix this problem, XSufferage adds a cluster-level sufferage value to each task, usually outperforming the conventional Sufferage in terms of the makespan of the resulting schedule.

Some of these list scheduling heuristics have been used in this work to provide a baseline for comparing the results achieved when using the proposed parallel evolutionary scheduling methods. Three of them have also been used to design probabilistic methods for the initialization procedure in the parallel EAs proposed in this work (the implementation details are presented in Chapter 5).

3.5.2 Metaheuristics for scheduling using ETC

In the last years, metaheuristic approaches have been frequently applied to find accurate schedules for HC systems. The next chapter presents a comprehensive review on the application of EAs to the HCSP and related problems; this subsection summarizes the most relevant works that have proposed applying non-evolutionary metaheuristics to solve the ETC-based model of the HCSP. The summary includes proposals using trajectory-based methods such as Simulated Annealing (SA), Tabu Search (TS), constructive techniques such as Greedy Randomized Adaptive Search Procedure (GRASP), swarm intelligence methods such as Ant Colony Optimization (ACO), and also fuzzy logic techniques and many hybrid algorithms, showing the diversity of metaheuristic applied to the HSCP.

Table 3.4 summarizes the most relevant works that have proposed using non-evolutionary metaheuristic methods to solve the HCSP and similar problems.

author(s)	year	algorithm	comment	problem size
Borriello, Miles	1994	SA	HC multiprocessors scheduling	60×6
Porto, Ribeiro	1995	ACO	precedence-constrained HCSP	up to 4000×10
Abraham et al.	2000	SA,TS,hybrids	HCSP and grid scheduling	N/D, theoretical
Braun et al.	2001	SA,TS	HCSP	512×16
Yarkhan, Dongarra	2002	SA	HC ScaLAPACK scheduling	12 and 25 machines
Blythe et al.	2005	GRASP	HC workflow scheduling	up to 1185×6
Onbaşioglu, Özdamar	2003	SA+hill climbing		N/D
Jakob et al.	2005	EA+LS	grid scheduling	87 tasks
Boyer	2005	randomized search	HCSP	up to 1000×5000
Stucky et al.	2007	EA+LS	grid scheduling	N/D
Singh, Bawa	2007	ACO+EA operators	grid scheduling	N/D
Ritchie, Levine	2004	ACO+TS	HCSP	512×16
Xhafa	2008	TS	HCSP	512×16

Table 3.4: Summary of metaheuristic methods applied to solve the HCSP.

Metaheuristics were excluded from the first comparative studies of HC scheduling techniques. When the size of the faced problem instances grew, researchers found themselves against the curse of dimensionality of NP-hard problems. Thus, emergent metaheuristic methods gained popularity to find accurate schedules in reasonable times.

The pioneering work from Borriello and Miles (1994) proposed a straightforward SA algorithm for HC multiprocessors scheduling, which used random moves for changing either the task-to-processor assignment or the task execution order on the same processor. Despite its simplicity, the method showed high efficacy when applied on an example problem with 60 tasks and 6 processors, achieving a high resource utilization factor (over 95%), while being fast enough for it to be applied in simulations for aircraft systems testing in use at Boeing. Another early work by Porto and Ribeiro (1995) applied TS to the HCSP with task precedence constraints. The initial solution was generated with a deterministic greedy heuristic and the candidate list of neighborhood solutions differs from the current one by a single task-to-processor assignment. Several attributes of the search history were considered to build the tabu list, while the aspiration criteria allowed identifying restrictions that may be override to improve the search. The TS approach was robust, and the results showed a makespan reduction of 20-30% over a greedy method for test problems ranging from 16 to 4000 tasks and up to 10 processors.

SA and TS were among the nature-inspired heuristics for scheduling on grid environments discussed in the generic description by Abraham et al. (2000). Both methods were aimed at optimizing makespan and flowtime using a steepest-descent/mildest-ascent approach for exploring the solution neighborhood, but the techniques were not further commented. Braun et al. (2001) included SA and TS in their comparison of methods for solving the HCSP. SA used a random task reassigning operator and an inverse exponential cooling scheme, while TS followed a two-stage exploration pattern involving a short hop to find the nearest local minimum and a long hop to move to unexplored regions of the search space. Both methods were unable to find accurate solutions, mainly due to the helpless search pattern for consistent scenarios. Later, YarKhan and Dongarra (2002) presented an analysis on applying a SA method integrated with grid services from the GrADS project Berman et al. (2001), for scheduling realistic instances of a numerical linear algebra routine (ScaLAPACK LU solver). Using information from an accurate performance model for the routine, SA explored the search space using a random add-remove-swap operator. SA outperformed an ad-hoc greedy scheduler on two HC clusters with 12 and 25 computers, by avoiding some local minima that were not anticipated by the greedy search.

Blythe et al. (2005) developed a GRASP scheduler for workflows in distributed HC environments, using a randomized Min-Min method to construct the solutions. The GRASP scheduler was compared with a traditional Min-Min method using a grid simulator built over ns-2, and real-life workflows of both compute-intensive and data-intensive applications. Using a time limit of 200 sec., GRASP was able to outperform Min-Min for scenarios with 57, 739, and 1185 tasks and 6 resources. The authors found that the workflow-based approach is less sensitive to uncertainty in ETC than a task-based approach. Using an improved version of Min-Min that explicitly accounts for idle times significantly speeded up the workflow-based approach, allowing tackling scenarios with thousands of tasks. Boyer and Hura (2005) presented RS, an iterated randomized method for scheduling dependent tasks in HC systems, based in an ad-hoc scheduling heuristic that executes in linear order with respect to the number of tasks and machines.

RS outperformed two GAs and a list scheduling technique for simulated scenarios modeled by large ETC matrices and real world parallel applications, but failed to find the most efficient schedules in consistent scenarios when using a 100 sec. stopping criterion. The article did not present numerical results for each scenario, it only reported the comparative performance of RS against GAs and the list scheduling technique.

The line of research on applying hybrid metaheuristics to the HCSP has shown that by exchanging a small set of partial solutions and/or statistical values in an efficient manner, hybrid schedulers could take advantage of combining methods to improve the search, outperforming single heuristics techniques.

The pioneering work on hybrid methods tackled simple variants of the HCSP. The hybridization of GA with SA and TS was suggested by Abraham et al. (2000), who claimed that the simple combination of methods has better convergence properties than a pure evolutionary search, although no experimental evidence was provided. Onbaşioglu and Özdamar (2003) presented a hybrid combining SA with hill climbing to minimize the total execution time of a parallel program. The authors reported satisfactory results when applying the hybrid method for solving two studied applications, one of them being computation intensive and the other being communication intensive.

Ritchie and Levine (2004) proposed an hybrid ACO for the HCSP. The ACO encoded in the pheromone trail the advantage of scheduling any given task onto each available processor, and each ant used a constructive heuristic method based on the (scaled) inverse MCT of tasks, which resembles a probabilistic Min-Min heuristic. Two local search (LS) methods were studied in order to improve the exploration: a fast LS method based on task move and swap operators, and a TS method specially designed to enhance the ACO search. The hybrid ACO+TS approach achieved the best makespan results, improving over the method combining ACO with Min-Min and LS. In addition, ACO+TS was able to achieve the best-know makespan results (at that time) for the set of HCSP instances proposed by Braun et al. (2001). However, the method took a long time (over 3.5 hours) to complete 1000 ACO iterations, thus it is not useful to perform on-line scheduling in dynamic HC and grid environments.

Jakob et al. (2005) presented a hybrid method combining an elitist EA that used problem-configurable encodings with application-independent LS methods. A mutation operator guided the search by performing small parameter changes and gene add-delete-reorder operations. Experiments later reported by Stucky et al. (2007) suggested that the method could be a promising tool for performing grid resource allocation. Recently, Singh and Bawa (2007) proposed a hybrid method for scheduling in grid environments, by using a fitness function that combines makespan and delays. The hybrid approach consists in an ACO improved by adding genetic operators that help to reduce the makespan, resource usage, and delay in meeting user specified deadlines, but no experimental analysis was provided.

Prior to the research reported in this thesis, the best results for the HCSP benchmark instances by Braun et al. (2001) were achieved by the TS scheduler proposed by Xhafa et al. (2008b). The problem model follows a hierarchic approach, considering makespan as a primary objective and flowtime a secondary one. The approach employed an integer-based encoding, Min-Min was used to generate the initial solution, and the TS movements were based on task moves and swaps. The tabu memory stored previous task-to-resource assignments and solutions already visited. Some elite solutions were kept during the evolution, and several aspiration criteria based on fitness, tabu list, and

local makespan were used. The search included an intensification phase to explore promising regions of the search space using the elite solutions and neighborhood structure changing, while soft and hard diversification methods were applied (using task distribution, penalizing ETC values, freezing tasks, etc.). Using a predefined stopping criterion of 100 sec., TS was able to improve over the previous best-known results –achieved using the hybrid ACO+TS by Ritchie and Levine– in ten of the twelve HCSP instances by Braun et al. (2001). In addition, experiments performed using a more realistic benchmark ranging from 32 to 256 machines showed that TS also outperformed a steady state GA for all the problem instances studied.

The results previously obtained with the ACO+TS by Ritchie and Levine (2004) and the TS by Xhafa et al. (2008b) –the two previous state-of-the-art methods for the HCSP– are used as a reference baseline to compare the results of the parallel EAs proposed in this work.

3.6 Summary

This chapter has presented the main general concepts about scheduling in heterogeneous computing environments, and the HCSP has been introduced and formally defined. Several considerations about the problem model used in this work have been commented in order to justify the relevance of the problem version tackled in distributed HC and grid environments. The chapter also described the key concepts about execution time estimation techniques, that are usually employed to estimate the execution time for a set of task in a heterogeneous system. The ETC estimation model has been presented, and the existing HCSP instances by Braun et al. (2001) have been commented. A brief survey about current grid and volunteer computing infrastructures has been included, which provides the motivation for the design of new large-sized HCSP instances that model large HC environments and medium-sized grid systems. The last section presented a brief review of the most popular deterministic heuristics and non-evolutionary metaheuristic approaches using execution time estimation, whose main ideas have been used for designing the EAs for the HCSP proposed in this work.

Chapter 4

Related work: EAs for HC scheduling

In the last fifteen years, the HCSP has become important due to the increasing popularity of parallel and distributed HC systems for cooperatively solving complex problems. Since the HCSP is NP-hard, heuristic and metaheuristic algorithms have been proposed in order to find accurate sub-optimal results in reasonable times. Among other metaheuristic techniques, EAs have been successfully applied for solving scheduling problems in heterogeneous environments, achieving the high level of problem solving efficacy also shown in many other areas of application.

This chapter presents a review of previous works that have proposed applying EAs to the static HCSP in the standard formulation given in Section 3.2, and also to solve related variants of the problem.

4.1 Introduction

EAs have been progressively applied to solve the HCSP and related scheduling problems since the middle of the 1990's decade. The analysis of related works allows identifying three periods, regarding the scope of the proposals:

- **1995-2000: HCSP in multiprocessors**. In this first stage the concept of a distributed grid of computational resources was incipient, so multiprocessor architectures were the focus of the pioneering works on applying metaheuristics to scheduling in HC environments. EAs emerged from those first proposals as a promising alternative to deterministic heuristics, in order to efficiently find accurate schedules for executing tasks in heterogeneous multiprocessors systems. However, the limited scalability of the multiprocessor parallel architecture put a practical upper bound on the number of resources involved in the scheduling problem, thus restricting to clearly demonstrate the advantages of EAs and other nature-inspired approaches for solving large instances of the scheduling problems.

- **2000-2005: distributed HCSP**. The new millennium saw the consolidation of distributed computing to harness the computer power resources in order to solve computing-intensive problems. The grid infrastructure started to consolidate, and as the size of HC systems increased, heuristics and metaheuristics appeared as the natural way to solve the related scheduling problems. Numerous researches proposed applying EAs to solve the HCSP on distributed environments in this period. Although many works followed the general concepts for HC scheduling proposed for multiprocessors in the previous stage, valuable new ideas were devised in order to fully exploit the search capabilities of EAs.

- **2005-2009: HCSP on grids**. Recently, the research community has focused on systematizing the application of EAs to the HCSP on grids, trying to provide compact and efficient methods for solving real-life scenarios. Several key issues such as techniques hybridization, incorporating specific problem knowledge, and using agent-driven approaches have been proposed to deal with the complexity of current and future heterogeneous multi-tiered grid infrastructures. Some methods have even been integrated into real-life tools for scheduling in HC and grid environments.

Figure 4.1 shows the three stages on applying EAs to the HCSP in a timeline from 1990 to the present time. The most relevant works from these three periods are summarized in the following sections.

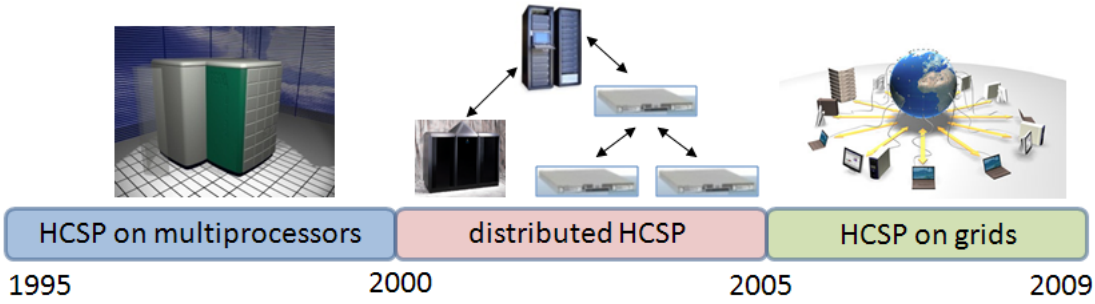


Figure 4.1: Timeline of EAs applied to the HCSP.

4.2 1995-2000: HSCP on multiprocessors

Back in the middle of the 1990's, the pioneering work on scheduling in multiprocessor computers provided a foundation for solving more complex problem instances and also scaling to large environments. The first proposals often faced small scenarios, tackling the problem of scheduling a limited number of dependent tasks, represented by a directed acyclic graph (DAG), a slightly different formulation than the one presented in Section 3.2. However, those seminal works provided the first hints on the applicability of EAs to solve scheduling problems in distributed HC systems. The most relevant works from this period are briefly summarized below.

The pioneering proposal by Tirat-Gefen and Parker (1996) dealt with the problem of designing heterogeneous multiprocessor systems and simultaneously minimizing the execution time of a given DAG. The approach combined a mixed integer linear programming model with a GA for finding near optimal designs of an application-specific multiprocessor. However, in the last twelve years, the most common approaches in HC scheduling considered the problem of minimizing some performance-related metric, usually the makespan, on a given machine suite.

Singh and Youssef (1996) applied a GA to the static mapping and scheduling of a set of tasks with dependencies on HC clusters. The experiments showed that the GA obtained high quality results in significantly reduced execution times, when compared with those required by polynomial-order deterministic heuristics for a set of randomly built DAGs with up to 100 nodes. Shroff et al. (1996) solved the HCSP using a hybrid Genetic Simulated Annealing (GSA) algorithm, conceived to retain the strengths of the two metaheuristic techniques combined. GSA was able to achieve the optimal values – computed by an exhaustive search of the solution space– for a set of randomly generated problem instances with up to 100 tasks and 35 hosts, although the required CPU times to compute the schedule significantly increased when solving large problem instances.

Wang et al. (1997) proposed applying a GA to solve the HCSP aimed at minimizing the makespan. This proposal started a specific line of research in HC scheduling, since many posterior works adopted their GA approach to solve diverse HCSP variants by using an ETC model. The problem formulation by Wang et al. (1997) split the mapping (allocation of tasks in machines) and scheduling (ordering of tasks execution in each machine), so a specific two-vector encoding was used, while it also assumed an estimated ETC for each task on each machine. The GA used a random initialization with a seeding procedure using deterministic heuristics in order to speed the search. The experimental analysis considered instances up to 100 tasks and 20 machines, where the GA was able to achieve better results than other non-evolutionary heuristics and a random search.

Kwok and Ahmad (1997) proposed Parallel Genetic Scheduling (PGS), a parallel GA for scheduling DAG-based applications on both homogeneous and heterogeneous computing environments. PGS followed a synchronous distributed subpopulation model using a migration operator that exchanged the best individual among subpopulations, considering a fully connected topology. The experimental results on scheduling random graphs and numerical algorithms with up to 500 tasks in a Intel Paragon with 16 processors showed that PGS outperformed the best list scheduling methods in terms of both solution quality and running time. In addition, PGS showed an almost linear speedup and suggested a good scalability behavior.

Grajcar (1999) faced the problem of mapping a partially ordered set of tasks to an HC multiprocessor system, trying to minimize the makespan while satisfying data dependencies and resource usage constraints. A GA was combined with a list scheduling method in a hybrid GA+LS algorithm, which follows a steady-state evolution model using several evolutionary operators in an integrated fashion. GA+LS found the optimal solution for problem instances with up to 96 tasks, while significantly reducing the runtimes required by exact methods. Later, Grajcar (2001) applied GA+LS in HC systems, pointing out that a major weakness comes from the lack of information about tasks and the environment that forces to take several assumptions, thus facing a restricted version of the HCSP. Two simple cases in which GA+LS fails to find the optimum were analyzed, suggesting that forecasting tools may be used to enhance the approach.

4.3 2000-2005: HCSP on distributed environments

In 2000, a new stage on applying EAs to the HCSP started. Abraham et al. (2000) studied the *novel* paradigm of grid computing, analyzing three nature-inspired heuristics for task scheduling. A conceptual formulation of the dynamic task scheduling into geographically distributed HC resources was presented, and the authors claimed that the environment complexity forces to use non-traditional methods to improve the search. So, they suggested using GA, SA, TS, and two hybrid algorithms aimed at minimizing the makespan and flowtime. Several encoding issues were analyzed, and two ordered task and resource list representation was proposed, which are to be dynamically updated through load profiling and forecasting. No experimental results were reported, but the work provided a detailed algorithmic description. This proposal was one of the first conceptual analysis of applying hybrid EAs to the grid scheduling problem.

Theys et al. (2001) presented three EAs for solving the HCSP. The first approach followed the previous work from Wang et al. (1997), and it was later extended in the works by Braun et al. (2001). The second approach corresponded to a problem-specific domain for a particular case of hardware platform, but the authors presented a methodology to perform online task rescheduling (in real time) using a GA off-line static mapping as a reference baseline. The third method explored the planning of meta-tasks in an HC environment considering task dependencies. Following this line of work, Braun et al. (2001) presented a systematic comparison of eleven mapping heuristics for the static HCSP. The analysis included two EAs based on the GA by Wang et al. (1997), adapted for scheduling independent tasks. The GA used a task-oriented encoding, and the fitness function evaluated the makespan of the schedule. Two methods were compared to perform the population initialization: a random initialization and a seeding procedure using Min-Min. The GA used elitism, proportional selection, SPX, and a random change mutation operator, all of them applied on a population of 200 individuals. Three different stopping criteria were studied: a predefined effort set in 1000 generations, and two stagnation criteria. In the hybrid Genetic Simulated Annealing (GSA) algorithm, the SA cooling procedure set a threshold that was used to decide whether an offspring survives (if its makespan is lower than the makespan of its parents plus the SA temperature value) or not. The results showed that both GA and GSA were able to consistently obtain the best makespan values for all types of HC scenarios studied, while they were able to take benefit of the Min-Min seeding procedure to significantly improve the search. Both EAs found the best results, based on calculated confidence intervals at 95%.

Zomaya and Teh (2001) faced a HSCP variant aimed at achieve load-balancing, maximizing the resource utilization and minimizing the makespan. A centralized GA was proposed to perform the dynamic task allocation, adopting a two-dimensional task-based encoding and initializing the population using a sliding-window technique. The GA used proportional selection, cycle crossover, swap mutation, and employs a predefined effort stopping criterion. The experimental analysis showed that GA significantly outperformed the First Fit heuristic and a random allocation scheme in terms of makespan and processor utilization for problem instances up to 1000 tasks and 50 processors. The GA worked better when the number of tasks increased, achieving almost full processor utilization. When increasing the number of processors, the GA makespan improved but the average processor utilization deteriorated, suggesting that load-balancing is harder for larger systems. The GA dynamic load-balancing scheduler was very effective from a practical point of view, specially when scheduling a large number of tasks.

Barada et al. (2001) faced the HCSP with coarse-grained tasks using a Simulated Evolution (SE) algorithm to minimize the makespan. SE is a generic iterative heuristic that emphasizes the behavioral relationships between parents and offspring (while traditional EAs focus in genetic relationships), by combining a EA schema with stochastic constructive perturbations to avoid local optima. SE uses problem-dependent techniques in the initialization, evaluation, and evolutionary operators. The authors did not present numerical results, but the graphics of evolution over randomly generated HCSP instances with 100 tasks and 20 machines showed that SE obtained mixed results when compared with the GA by Wang et al. (1997), so the conclusions were not definitive.

Dhodhi et al. (2002) studied the problem of efficient scheduling DAGs onto a distributed HC system, and proposed a problem-space GA (PSGA) including a known fast problem-specific heuristic. PSGA was executed to minimize the schedule makespan using a task-priority based encoding, which is used by problem-specific decoding heuristics, while 2PX, elitism, and a perturbation mutation were applied in the problem space. Several test suites of well-known applications and randomly generated graphs up to 200 tasks and 20 processors were employed in the experimental evaluation. Using a predefined effort stopping criterion, PSGA obtained better makespan results than three other scheduling methods: a deterministic heuristic, GSA and the GA by Wang et al. (1997), while significantly improved the required execution time for computing the best schedule.

Scheduling tasks on an HC grid system in case of resources contention was the main focus of the works by Di Martino and Mililotti. A GA scheduler was proposed for maximizing the resources throughput while resolving conflicts in the power usage. The GA used task information provided by the users, and it operated in two levels: allocating a single task at a time on a local resource using local information, and using a centralized *superscheduler* GA to resolve the interaction with geographically distributed HC entities using HC hardware. The authors employed ad-hoc crossover and mutation operators to avoid infeasible solutions, and a weighted fitness function to compute the local makespan for each resource, trying to achieve a good load-balancing pattern. The GA obtained accurate solutions with limited resource power waste for a test case of 24 tasks and 6 resources (DiMartino and Mililotti, 2002). However, it failed to find the optimal solution for a test case with 4 resources and 32 tasks in 600 executions (DiMartino and Mililotti, 2004), when using a one-minute time limit stopping criterion, suggesting that a more adequate fitness function should be used.

Wu et al. (2004) presented an original GA that required minimal problem information for scheduling DAGs in a multiprocessor system. The GA admitted non-feasible solutions in the population, and avoided using problem specific operators or repair mechanisms to correct individuals. A problem encoding based on a list of task-processor pairs was proposed as a way to help the GA to identify and maintain tightly linked building blocks. The fitness function focuses on both ensuring the feasibility of tasks execution and minimizing the makespan. The GA followed an *incremental* approach that gradually increases the difficulty of fitness values until finding adequate schedules. Even though the work was mainly focused on solving the homogeneous multiprocessor scheduling problem, the authors performed experiments on a small HC environment with only four processors. The GA outperformed traditional scheduling methods, but since it needed a large population size to achieve the best results, the scalability of the approach to large problems is compromised.

Following the GA scheduler by Zomaya and Teh (2001), Page and Naughton (2005) proposed a dynamic micro-GA scheduler for HC environments. The micro-GA used a specific heuristic to create the initial population, while cycle crossover and two types of random swap and balancing mutations were applied. The evaluation was performed using a randomly generated test set of HCSP instances with up to 50 processors and up to 10000 tasks. Using a predefined effort stopping criterion, micro-GA performed better than several well known heuristics and the GA by Zomaya and Teh (2001), consistently achieving lower makespan values and higher processor utilization.

4.4 2005-2009: Heterogeneous grid scheduling

Last years have seen many proposals devoted to solve real-life HCSP instances with increasing dimensions, while also integrating the EA schedulers with high-level tools for grid environments.

In the context of the ASKALON project, Prodan and Fahringer (2005) presented ZENTURIO, a scheduling tool that combines GA with heuristics for grid environments, employing prediction models to compute high performance metrics of tasks. The GA used a workflow-parameter encoding to represent grid resources, fitness scaling, SPX, and random mutation. High quality results were reported for experiments using ZENTURIO to schedule a real-world scientific workflow application executing in a grid with over 300 machines.

The makespan robustness of HC schedules was investigated by Sugavanam et al. (2005). In the problem model, a system is defined to be robust if the makespan under perturbations in ETC estimates does not exceed a required time constraint. The studied methods included a steady state GA (SSGA), a Memetic Algorithm (MA) including a hill climbing swapping operator, and the HereBoy EA, a trajectory-based hybrid method combining GA and SA to accept worse solutions than the current one. All three EAs used an integer encoding, elitism, seeded initialization, and rank-based selection. The EAs were evaluated using a simulated HC system with 8 machines and 1024 independent tasks, using a predefined effort stopping criterion. Both SSGA and MA achieved the best makespan and robustness metric results, while HereBoy performed the worst. The methods were later applied to identify the hardware to build an HC cluster whose total cost falls within a specified budget, and simultaneously maximizing the makespan robustness (Sugavanam et al., 2007). The EAs were compared with three specific heuristics, running for 3000 s. to solve several HC scenarios with 1024 tasks. While GA and MA performed comparably to the heuristics, HereBoy achieved poorer results. Improved methods were proposed by combining EAs and specific heuristics, but the best exploration pattern was achieved using a technique that employs two GAs: one for selecting the set of machines and the other for later performing the task mapping.

In the last years, multiple works from Xhafa et al. have explored several variants of EAs applied to the HCSP.

Carretero and Xhafa (2006; 2007b) studied GAs for scheduling tasks in large scale grids. Two GA were presented: a hierarchical GA which in a first phase optimizes the makespan and next optimizes the flowtime, and a GA which optimizes both objectives simultaneously, using a linear aggregation fitness function. The authors analyzed using integer vector and permutation-based representations, and studied several variations for the population initialization and GA operators. An exhaustive experimental analysis

was reported in order to identify the best operators and fine tuning its parameters for static HCSP instances following the ETC model from Ali et al. (2000). In addition, a simple grid simulator was developed in order to deal with realistic large-scale HCSP instances, up to 4096 tasks and 256 hosts. The experimental results showed that the GAs were able to achieve fast makespan reductions for both the static and dynamic problem versions, showing the practical interest of GA-based schedulers for grid environments.

Duran and Xhafa (2006) solved the HCSP using the Struggle GA (SGA) (Grüninger and Wallace, 1997), an EA aimed at delaying the convergence by applying a generation gap model and a niche technique to maintain diversity. The experimental study compared SGA and a SSGA for the HCSP in grid resources, using a fitness function that combines makespan and flowtime using a linear aggregation. The EAs was evaluated using the HCSP instances from Braun et al. (2001) and random dynamic instances from the previous work. Both EAs were able to outperform the makespan results by Braun et al. (2001) in more than half of the instances studied. SSGA obtained better flowtime results for all but one instance, while SGA was able to achieve better makespan results for small instances, but requiring larger execution time than SSGA. For high dimension problem instances, the population diversity of SGA conspired against achieving high makespan reductions within an execution time of 90 s., suggesting that SSGA is a more adequate scheduler for dynamic grid environments.

Later, Xhafa et al. (2007a; 2008c) tuned the replacement mechanism, trying to reduce the computational cost of SGA. The authors introduced a new hash-based similarity measure for finding similar individuals on the population in constant time. Three struggle operators based on different hash keys were evaluated regarding the makespan metric for the HCSP instances from Braun et al. The SGA using a task-resource allocation hash key outperformed the previous (quadratic order) SGA implementation, while showing linear scalability, so the population size can be increased without downgrading the performance.

The line of work on applying MAs to the HCSP was summarized in the book chapter by Xhafa (2007), who also presented a MA for minimizing the makespan and flowtime. The MA used a task-based encoding and subordinate LS operators to provide high quality solutions in a very short time (a critical issue in dynamic grid environments). Several methods were considered to initialize the population, and a large set of LS operators were evaluated, ranging from simple local move and swaps to complex operators such as a TS method. The experimental analysis using the HCSP instances from Braun et al. (2001) and a 90 s. stopping criterion showed that the MA+TS hybrid performs better than previous GAs. Four scenarios up to 2096 tasks and 256 machines were used for the dynamic HCSP, where MA+TS achieved better makespan results than MA but worse flowtime results. Later, Xhafa and Duran (2008) presented parallel MA implementations, trying to reduce the large search times for the HCSP. The numerical experiments using fine-grained and coarse-grained versions of MA+TS showed that the parallel models were suitable to achieve several performance requirement on HC and grid scheduling.

Xhafa et al. (2008a) also explored the efficacy of using a cellular MA (cMA) for the HCSP, in order to efficiently exploiting the large amount of grid computing resources. The structured population gives cMA the ability of controlling the tradeoff between the exploitation and exploration of the solution space, helping to achieve high quality solutions in a very short time. The proposed cMA population was arranged using

a two-dimensional toroidal grid topology, several well-known neighborhood patterns were considered, and an asynchronous cell updating was used. cMA used a seeding initialization, SPX, a rebalancing mutation operator and three LS methods. The cellular model outperformed previous GA results for half the HCSP instances from Braun et al. (2001), while showing a robust behavior, able to achieve high quality planning in short times.

Following a slightly different line of work, Iordache et al. (2007) studied the real-time HCSP in grid environments. The author proposed a dynamic GA scheduler, able to efficiently execute in a fully decentralized system by using monitoring information to perform the task allocation. The GA used SPX and a task-moving mutation with adaptive probabilities, while the monitoring information was employed in the initialization, in the fitness evaluation, and also when receiving a new group of tasks. Several GA versions were evaluated on small real-time environments up to 12 processors and 100 typical CPU-intensive tasks, using true monitoring and execution systems. A distributed cooperative GA achieved the best load balancing and processor utilization results, while a non-cooperative GA was able to exploit multiple initial search points. Although the problem instances faced were too small, the work had a valuable contribution as it used a real environment and true monitoring and task execution tools. Iordache et al. (2007) also implemented a distributed and scalable GA scheduler for HC environments (SAGA), working in cooperation with grid services to overcome the lack of robustness of centralized GAs in realistic scenarios. SAGA clearly outperformed other GAs regarding load-balancing, processor utilization, as well as makespan minimization in an eleven node grid, and its convergence speed improved when the number of agents increased. The main ideas from the previous works were applied to design an integrated GA-based scheduling tool for the GridMOSI project (Neagu et al., 2007).

A recent work by Braun et al. (2008) tackled a general HCSP formulation concerning dependencies, priorities, deadlines and versions. The experimental analysis compared two greedy heuristics, a GA, and a steady-state GA with customized encodings and operators to solve scenarios with up to eight machines and 2000 tasks. The results showed that the steady-state GA performed the best among the studied heuristics, achieving the best schedules, whose fitness values were between 65% and 95% of upper bounds calculated under unrealistic assumptions.

4.5 Summary

The analysis of the related bibliography shows the large diversity of proposals on applying EAs for solving the HCSP, its related variants, and similar scheduling problems on HC environments. Since the first proposals in the early 1990s, much effort has been done on conceptualizing the HCSP, and EAs have been employed in order to exploit the exploration pattern of the evolutionary search for finding accurate results. However, when high levels of computational efficiency are required, the traditional formulation of EAs seems to be rather slow for achieving high quality results within a few minutes of execution time. Trying to overcome this problem, researchers have often combined EAs with specific heuristics in the quest for obtaining accurate results. Table 4.1 summarizes the most relevant works on applying EAs to the HSCP and related variants.

Period	Author(s)	Year(s)	Method(s)
1995-2000:	Tirat-Gefen, Parker	1996	MEGA (GA+LP)
	Singh, Youssef	1996	GA
HCSP on	Shroff et al.	1996	GSA
	Wang et al.	1997	GA
multiprocessors	Kwok, Ahmad	1997	GA
	Grajcar	1999, 2001	hybrid GA+LS
	Abraham et al.	2000	GA, GASA, GATS
	Theys et al.	2001	EA
2000-2005:	Ali et al.	2000	GA
	Braun et al.	2001	GA, GASA
distributed	Zomaya, Teh	2001	dynamic GA
	Barada et al.	2001	simulated evolution
HCSP	Dhodhi et al.	2002	problem-space GA
	Di Martino, Mililotti	2002, 2004	two-level GA
	Wu et al.	2004	incremental GA
	Page, Naughton	2005	micro-GA
	Prodan, Fahringer	2005	GA + heuristics
	Sugavanam et al.	2005, 2007	GA, MA, HereBoy, hybrids
2005-2009:	Carretero, Xhafa	2006, 2007	GA
	Duran, Xhafa	2006	Struggle GA
HCSP on	Xhafa et al.	2007, 2008	GA
	Xhafa et al.	2007, 2008	MA, MA+TS
grids	Xhafa et al.	2007	cMA
	Iordache et al.	2007	multiagent GA
	Braun et al.	2008	GA

Table 4.1: EAs applied to the HSCP.

The first works allowed identifying EAs as promising methods for solving scheduling problems on heterogeneous multiprocessors. Most of the proposals faced DAG-based applications with only few tasks and resources, but seminal works such as the one by Wang et al. (1997) provided a foundation for more complex methods later proposed following this approach. Due to the limited scalability of the multiprocessor architecture, the experimental evaluation was often performed using few processors, and EAs had few opportunities to show their potentiality for solving complex scheduling instances.

When distributed computing emerged as an important resource for solving complex problems with high computing demands, the relevance of the HCSP also raised. Since the first works in the 2000's decade, the researchers showed a renewed interest for designing accurate schedulers, able to deal with problems involving an increasing number of tasks and machines. The independent task approach was adopted since it is more accurate to model realistic situations in large HC and grid environments. The consensus formulation for the HCSP aiming at minimizing the makespan dates from the early works by Ali et al. (2000) and Braun et al. (2001), who also presented the first proposal of a benchmark test suite constructed following a specific methodology. In addition, the works by Zomaya and Teh (2001) suggested many GA and EA variants in order to provide efficient HCSP solutions, while the first proposals of parallel-distributed GAs such as the one presented in the works by DiMartino and Mililotti (2002, 2004) was also formulated. The proto-implementation of a micro GA scheduler by Page and Naughton (2005), which is specially relevant for the approach adopted in this work, also dates from this period.

Recently, the line of research by Xhafa et al. systematized the empirical evaluation of EAs applied to the HSCP, which also had a great influence in this work. Xhafa et al. analyzed many EA variants, focusing in traditional and steady-state GAs, MAs, and the cellular model. They also provided the first experiments on solving HCSP instances of realistic large-size instances, using LS methods in order to improve the efficiency and the efficacy of the evolutionary search. The best-known results for the benchmark problems by Braun et al. (2001) were achieved following this line of work. Many other key issues such as hybridization, incorporating specific problem knowledge, and using agent-driven approaches have been proposed by other authors in order to deal with the HCSP complexity. The research community have also diversified the problem approaches, studying novel aspects such as the makespan robustness, the stochastic nature of execution times, economic-driven methods, and other high-level issues for achieving a scalable and highly reliable scheduling tool.

Despite the numerous proposals on applying EAs to the HCSP and related scheduling problems, there have been few works studying large-size and realistic HCSP instances in grid environments, mainly due to the inherent complexity of dealing with the underlying high-dimension optimization problem. In addition, few works have studied parallel algorithms, in order to determine their ability to use the computing power of large clusters to improve the search. The survey of related works also allowed to conclude that there do not exist standardized problem benchmarks or test suites –except the low-dimension, de-facto standard problems by Braun et al. (2001)–. Thus, there is still room to contribute in those lines of research, by studying highly efficient parallel EA implementations, able to deal with large-size HCSP instances by using the computational power of parallel and distributed environments, as it is presented in this thesis.

Chapter 5

Parallel evolutionary algorithms for the HCSP

The previous chapter summarized the most relevant related works on applying EAs and other evolutionary methods to the HCSP. Even though the review showed some proposals on developing parallel models for improving the computational efficiency of the studied methods, there have been few references about using PEAs –or other parallel metaheuristics– for solving large HCSP instances in order to model realistic HC and grid environments. This chapter presents the implementation details of three EAs applied to the HCSP: a traditional GA, and a CHC algorithm, both of them in their sequential and parallel variants, and a new parallel micro-CHC algorithm specifically developed in this work.

All three EAs were designed trying to achieve accurate solutions in reduced time, while providing a good exploration pattern that allows to efficiently solve large-size HCSP instances. To achieve these goals, several techniques have been applied to reduce the execution time of the studied methods: two alternative encodings were proposed, as well as several options for population initialization, and many variants of evolutionary operators were analyzed. The details about these alternatives and variants are presented in this chapter, along with the software library in which the EAs were implemented.

5.1 The MALLBA library

The MALLBA project (Alba et al., 2002) is an effort to develop a library of algorithms for optimization that can deal with parallelism (on a Local Area Network (LAN) or on a Wide Area Network (WAN)), in a user-friendly and, at the same time, efficient manner. The EAs described in this chapter are implemented as generic templates on the library as *software skeletons*, to be instantiated with the features of the problem by the user. These templates incorporate all the knowledge related to the resolution method, its interactions with the problem, and the considerations about the parallel implementations. Skeletons are implemented by a set of *required* and *provided* C++ classes that represent an abstraction of the entities participating in the resolution method:

- The **provided classes** implement internal aspects of the skeleton in a problem-independent way. The most important *provided* classes are `Solver`, which abstracts the selected resolution method and `SetUpParams` that contains the setup

parameters needed to perform the execution (e.g. number of iterations or generations, number of independent runs, parameters guiding the search, etc.).

- The **required classes** specify information related to the problem. Each skeleton includes the `Problem` and `Solution` required classes, that encapsulate the problem-dependent entities needed by the resolution method. The `Problem` class abstracts the features of the problem that are relevant to the selected optimization method. The `Solution` class abstracts the features of the feasible solutions that are relevant to the selected resolution method. Depending on the skeleton, other classes may be required.

The conceptual separation between classes allows defining required classes with a fixed interface but without supplying specific implementations, thus provided classes can use required classes in a generic manner. The `Solver` class provides methods to run the resolution algorithm and methods to consult its progress or change its state. The only information the solver needs is an instance of the problem to solve and the setup parameters. In order to enable a skeleton to have different solver engines, the `Solver` class defines a unique interface and includes several subclasses that supply different sequential and parallel implementations (`Solver_Seq`, `Solver_Lan` and `Solver_Wan`). Figure 5.1 presents a UML diagram showing the MALLBA classes and their interactions.

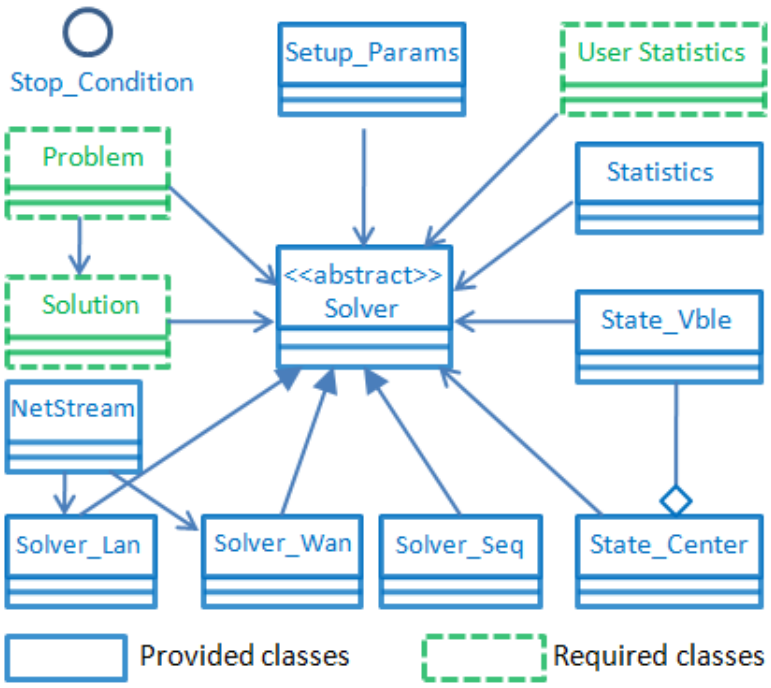


Figure 5.1: UML for MALLBA classes.

The MALLBA library is publicly available to download at University of Málaga website `http://neo.lcc.uma.es/mallba/easy-mallba`. Using this library has allowed a quick coding of different algorithmic prototypes to cope with the inherent difficulties of the HCSP.

The implementation of pμ-CHC is based on the CHC skeleton provided by MALLBA. Additional code was incorporated to define and manage the external population (including the implementation of a remove-of-the-worst strategy when inserting new individuals into the elite set), to implement the specialized reinitialization and local search operators proposed for the micro population in pμ-CHC, and to include other features related to the HCSP resolution. The details about the problem encoding, the implementation of the evolutionary operators, and specific features of each EA are provided in the next sections.

5.2 Problem encoding

Two main alternatives have been proposed in the related literature for encoding HCSP solutions when dealing with independent tasks: the *task oriented* encoding and the *machine oriented* encoding.

The task oriented encoding uses a vector of machine identifiers to represent the task-to-resource assignment, as it is presented in Figure 5.2. The one-dimension vector has N elements, where the presence of m_j in the position t_i means that the task t_i is scheduled to execute on machine m_j. The task oriented encoding is a direct representation for schedules that has been frequently used in related works, since it allows a straightforward exploration by using simple operators based on moving and swapping task assignments. However, after applying a move or swap operator, the task oriented encoding does not provide an easy way to evaluate the changes on efficiency metrics related to the whole schedule (such as makespan, flowtime or resource utilization). When using the task oriented encoding, any single change on a task assignment forces to reevaluate the schedule metric.

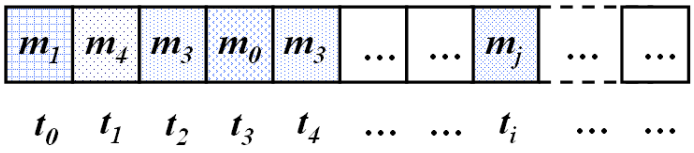


Figure 5.2: Task oriented encoding.

The machine oriented encoding uses a two-dimensional structure in order to represent the group of tasks scheduled to execute on each machine m_j. Figure 5.3 presents an example of the machine oriented encoding, showing for each machine m_j the list of tasks t_k assigned to it. The machine oriented encoding provides an easy and efficient way for performing exploration operators based on moving and swapping task assignments, since it is able to store specific values of efficiency metrics for each machine (such as the local makespan). Thus, the makespan variation when performing changes on task assignments can be efficiently calculated considering only the tasks and machines involved in the move or swap performed, without requiring to reevaluate the efficiency metric for the whole schedule.

Both encodings were applied in the EAs studied in this work. In the prototype implementations of traditional GA and CHC methods, both serial and parallel versions used the task-oriented encoding, in order to provide a simple method for analyzing the efficiency of EAs for solving the HCSP. In a second stage, the machine-oriented encoding was adopted, trying to improve the efficiency of the makespan calculation.

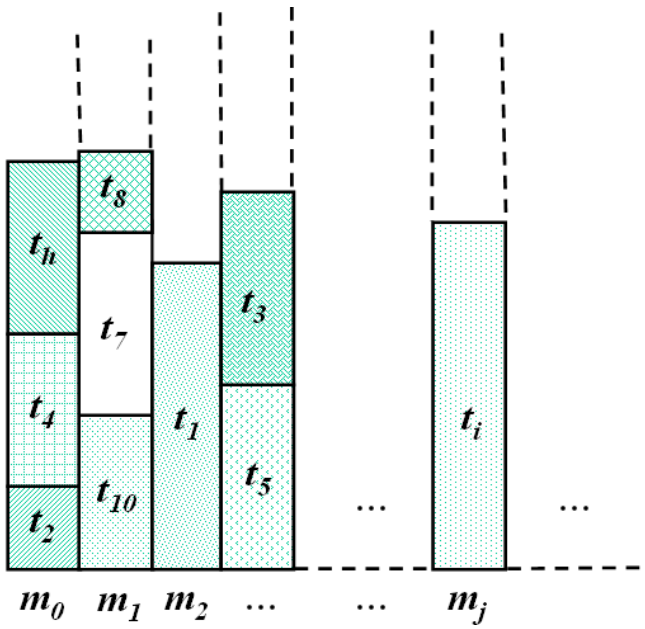


Figure 5.3: Machine oriented encoding.

Those improved GA and CHC versions allowed to store the local makespan values, significantly enhancing the computational efficiency of the search.

The parallel GA and CHC versions, and also the new pμ-CHC uses the machine-oriented encoding, since it provides the best efficiency for speeding up the evolutionary search, allowing tackling large dimension HCSP instances.

5.3 Fitness function

The studied version of the HCSP considers the makespan as the objective function to minimize (see Section 3.2). Since the standard formulation of EAs assumes a fitness function to maximize, as it was presented in Chapter 2, the proposed algorithms considered the opposite value of the makespan metric as the fitness function (i.e. $fitness = makespan \times (-1)$). This issue is internally handled by the MALLBA library, which solves a minimization problem if the sentence `direction = minimize` is included in the required code for the correspondent algorithmic skeleton. The fitness function takes positive real values.

5.4 Population initialization

Numerous methods have been proposed to generate the initial population in the related works on applying EAs to the HCSP (Braun et al., 2001; Xhafa and Duran, 2008; Xhafa et al., 2008a). Many of those proposals employed specific ad-hoc heuristics with the objective of starting the evolutionary search from a set of useful suboptimal schedules, increasing the EA effectiveness to minimize several efficiency metrics. Besides simple random initialization operators, scheduling heuristics such as Min-Min, SJFR, LJFR, MCT, and others, have often been applied with diverse effectiveness when minimizing makespan and flowtime in the HCSP.

In this work, several methods were studied to generate accurate initial solutions for the EA population in order to speed up the search. When dealing with low-dimension HCSP instances, deterministic heuristics such as Min-Min and Sufferage provide accurate and easy-to-compute solutions to seed the population. The Min-Min strategy has been identified as an efficient method for finding accurate schedules for small size HCSP instances (Braun et al., 2001), and also when the ETC matrix has reasonable variations on task and machine heterogeneity (Luo et al., 2007), while Sufferage often achieved better schedules than Min-Min for inconsistent scenarios.

However, when the problem dimension grows, the time required to compute the initial solution increases, thus reducing the EA efficiency. To avoid the performance degradation, probabilistic versions of Min-Min and Sufferage heuristics have been used in this work for the population initialization when solving large HCSP instances. The probabilistic versions follow the general procedure of the correspondent deterministic heuristic, but only for assigning a random number of MAX_{init} tasks, while the remaining tasks are assigned using a MCT strategy.

5.5 Evolutionary operators

This subsection present the implementation details of the evolutionary operators, describing the recombination, mutation, reinitialization and local search used in GA, CHC and pμ-CHC.

5.5.1 Exploitation: recombination

The classic GA uses the Single Point Crossover (SPX) operator to recombine the characteristics of two solutions. SPX is directly applied when using the task oriented encoding, since cutting two schedule encodings and swapping the corresponding alleles (tasks-to-machine assignments) from one parent to another always produces feasible solutions. When using the machine-oriented encoding, SPX works selecting a cutting point and after that each task from one parent is swapped to the corresponding machine in the other parent.

CHC and pμ-CHC uses HUX to recombine characteristics of two solutions. When using the task oriented encoding, the HUX implementation is straightforward: for each task, the corresponding machine in each offspring is chosen with uniform probability between the two machines for that task on the parents' encoding. When using the machine-oriented encoding, each task from one parent is swapped to the corresponding machine in the other parent with a probability of 0.5.

5.5.2 Exploration: mutation and reinitialization

Both the mutation (used in GA) and reinitialization (used in CHC and pμ-CHC) are random operators that perform small perturbations in a given schedule, aimed at providing diversity to the population, to avoid the search from getting stuck in local optima. These operators performs simple moves and swaps of tasks between two machines, selecting with high probability the machines with highest and lowest local makespan (`heavy` and `light`, respectively).

The GA applies the mutation operator on selected individuals along the evolutionary search, while CHC applies the reinitialization using the best individual found so far as a template for creating a new population after convergence is detected.

The mutation and reinitialization operators cyclically perform a maximum number of `MAX_TRIALS` move-and-swap task operators, which include:

1. Move a randomly selected task (selecting the longest task with a probability of 0.5) from `heavy` to `light`.
2. Move the longest task from `heavy` to the most suitable machine (the machine which executes that task in minimum time).
3. Move into `light` the best task (the task with the lowest execution time for that machine).
4. Select a task from `heavy` (selecting the longest task with a probability of 0.5), then search the best machine to move it to, regarding the current schedule.

Each time that a task is moved from a source machine to a destination machine, a swap movement is randomly applied with a probability of 0.5, moving a different task from the destination machine to the source machine. Unlike previous exploration operators for the HCSP presented in related works by Xhafa et al. (2006; 2007b; 2008b), none of the foregoing operators imply exploring the $O(n^2)$ possible swaps, not even exploring the $O(n)$ possible task movements. The four exploration operators used in the proposed EAs are performed in sub-linear complexity order with respect to both the number of tasks and the number of machines in each HCSP instance. This feature allows the proposed EAs to show a good scalability behavior when applied to solve large HCSP instances that model large HC and grid systems.

The pseudo-code of the mutation and reinitialization operators for the HCSP is presented in Algorithm 6. Trying to additionally improve the resulting schedules, a rebalancing operator is randomly applied with a probability of 0.5 after applying the mutation operator. This operator was designed in order to fill the gap on the local makespan between the heavy machine and the light machine, by performing task moves between them while it is possible, trying to balance the load in the schedule.

Algorithm 6 HCSP mutation and reinitialization.

```
    Input: schedule s
 1: select machines HM y LM
    {select heavy and light machines with probability HEAVY_MACH}
 2: trials ← 0
 3: end_search ← FALSE
 4: orig_makespan ← makespan(s)
 5: repeat
 6:    select mut_operator
 7:    if mut_operator == 1 then
 8:       move a randomly selected task from HM to LM
          {with a probability of 0.5 move the longest task assigned to HM}
 9:    else if mut_operator == 2 then
10:       move the longest task from HM to the suitable machine
11:    else if mut_operator == 3 then
12:       move the best task to LM
13:    else
14:       search the machine that minimizes the MCT for a randomly selected task from HM
          {select the longest task with a probability of 0.5}
15:    end if
16:    apply swap from destination to source (with probability 0.5)
17:    trials ← trials + 1
18:    if makespan(s) < orig_makespan then
          {Makespan improvement: end the cycle}
19:       end_search ← TRUE
20:    end if
21: until ((trials == MAX_TRIALS) OR (end_search))
22: apply rebalancing operator (with probability 0.5)
```

5.6 Local search: randomized PALS

Many strategies have been proposed in the related literature for providing diversity and improving the efficacy of the search when using EAs for solving the HCSP. Most of the previous works concluded that local search methods are needed within any EA to find accurate schedules in reasonable short times. The works of Xhafa et al. (2007b; 2008a; 2008b) explored several local search operators for solving low-dimension HCSP instances, but many of the proposed operators become ineffective when the problem instances grow. Initial experiments showed that the reinitialization operator used in the traditional CHC algorithm did not provide enough diversity when working with small subpopulations, thus restraining the parallel algorithm to work using eight demes with 15 individuals each. In order to improve the population diversity, the pμ-CHC algorithm incorporates a randomized version of a well-known local search method, already employed with success for solving hard combinatorial optimization problems.

The local search operator used in the pμ-CHC algorithm is based on Problem Aware Local Search (PALS), a novel heuristic algorithm originally proposed for the DNA fragment assembly problem (Alba and Luque, 2007). PALS is based on the classic 2-opt heuristic, one of the most famous methods in the class of local search algorithms known as *exchange heuristics* (because they are based on simple tour modifications) for solving the Traveling Salesman Problem (TSP).

2-opt uses a move that deletes two edges on a given cycle, thus splitting the tour into two paths, and then reconnects those paths in the other possible way. Even though the basic concepts on using the breaking-and-reconnecting moves had been proposed earlier, the 2-opt heuristic was first formulated by Croes (1958). Since then, 2-opt has been used as the building block for many algorithms for solving the TSP, most notable in the TS-like method by Lin and Kernighan (1973) (which in fact was proposed more than ten years before Glover conceptualized the TS metaheuristic approach).

PALS extended the original 2-opt concepts to design a specific search method for solving discrete optimization problems. The general specification of PALS by Alba and Luque (2007) is presented in Algorithm 7.

Algorithm 7 Generic schema of PALS method.

```
   Input: solution s
 1: repeat
 2:    L ← ∅
 3:    for i = 0 to N do
 4:       for j = 0 to N do
 5:          Δ_f ← calculateDelta(s, i, j)
 6:          if Δ_f > 0 then
 7:             L = L ∪ ⟨i, j, Δ_f⟩ {Add candidate movements to L}
 8:          end if
 9:       end for
10:    end for
11:    if L ≠ ∅ then
12:       ⟨i, j, Δ_f⟩ ← Select(L) {Select a movement among candidates}
13:       applyMovement(s, i, j) {Modify the solution}
14:    end if
15: until no changes on s
```

PALS works on a single solution s, which is iteratively modified by applying a series of movements aimed to locally improve their function value $f(s)$. The movement operator performs a modification on the positions i and j in a given solution s, while the key step is the calculation of the objective function variation Δ_f when applying a certain movement. When the calculation of Δ_f can be performed without significantly increasing the computational requirements, PALS provides a very efficient search pattern for combinatorial optimization problems.

Following the generic schema presented in Algorithm 7, a specific variant of PALS was designed for the HCSP. The method is devoted to exploring possible task swappings between machines in a given schedule, trying to improve the makespan metric. However, due to both the huge dimension of the search space, specially when solving large HCSP instances, and the goal of achieving accurate results in short execution times, the deterministic paradigm in PALS was replaced by a randomized one (i.e. the local search uses random criteria to define the set of adjacent swaps explored). The randomized version also incorporates two other differences with the generic PALS algorithm: the main cycle ends when finding a solution that improves the schedule makespan, and if the search does not find an improved solution, `MAX_TRIALS` attempts are performed applying the swap that produces the lowest makespan degradation, trying to introduce diversity in the EA population.

Algorithm 8 presents the pseudo-code of the randomized PALS for the HCSP. Working on a given schedule s, the randomized PALS selects a machine m to perform the search. With high probability (`HEAVY_MACH`) the machine with the largest local makespan is selected, focusing on improving the assignment for the machine which defines the makespan of the whole schedule, but also introducing a chance of improving the local makespan for other machines. The outer cycle iterates on `TOP_M` tasks assigned to machine m (randomly starting in task `start_m`), while the inner cycle iterates on `TOP_T` tasks assigned to other machines (randomly starting in task `start_t`). For each pair (t_M, t_T), the double cycle calculates the makespan variation when swapping tasks t_M and t_T. The method stores the best improvement on the makespan value for the whole schedule found in the `TOP_M` × `TOP_T` swaps evaluated. After the double cycle ends, the `best_move` found so far is applied, disregarding whether it produces an effective makespan reduction or not. The process is applied until finding a schedule which improves the original makespan or after performing `MAXIT_PALS` attempts.

Algorithm 8 Randomized PALS for the HSCP.

```
    Input: schedule s
    Select machine m {select heavy with probability HEAVY_MACH}
 1: trials ← 0; end_search ← FALSE
 2: orig_makespan ← Makespan(s)
 3: repeat
 4:    Δ_BEST ← ∞
 5:    for t_M = start_m to TOP_M do
          {Iterate on tasks of machine m}
 6:       for t_T = start_t to TOP_T do
             {Iterate on tasks of other machines}
 7:          Δ_M ← calculateDeltaMakespan(s, t_M, t_T)
 8:          if Δ_M < Δ_BEST then
 9:             best_move ← ⟨t_M, t_T, Δ_M⟩ {Store best move found so far}
10:             Δ_BEST ← Δ_M
11:          end if
12:       end for
13:   end for
14:   trials ← trials + 1
15:   applyMovement(Best_move) {Modify the solution}
16:   if Makespan(s) < orig_makespan then
          {Makespan improvement: end the cycle}
17:       end_search ← TRUE
18:   end if
19: until ((trials == MAXIT_PALS) AND (! end_search))
```

The randomized version of PALS was designed to provide a powerful search pattern for the HCSP. It allows moving toward local optima in the space of HCSP solutions each time that an improved solution is found, and it also provides diversity to solutions –allowing the algorithm to scape from strong local optima– after applying `MAXIT_PALS` changes when no improved solution is found. The calculation of the makespan variation when swapping two tasks (calculateDeltaMakespan(s, t_M, t_T)) is performed without requiring high computational requirements when using the machine oriented encoding, since it stores the local makespan of each machine. Thus, PALS provides a very efficient search pattern for the HCSP in pμ-CHC.

In addition, the PALS operator provides a generic search pattern for pμ-CHC, a feature seldom found in other existing local search algorithms for the HCSP, which allows the new algorithm to be applied to other similar optimization problems.

5.7 Speeding up the pμ-CHC search

This section presents the main features of pμ-CHC, conceived to speed up the evolutionary search in order to reach accurate results in short execution times for hard-to-solve optimization problems. The first subsection comments the main characteristics of the accelerated convergence evolution model, and the last subsection describes the two-level parallel model employed in the pμ-CHC implementation.

5.7.1 Accelerated convergence in pμ-CHC

Diversity is quickly lost when using low population sizes. In order to speed up the convergence of the pμ-CHC algorithm, the PALS operator is applied after a certain (low) number of generations pass without inserting any offspring into the new population during the mating procedure. This accelerated cataclysmic model showed the ability of providing enough diversity to avoid the EA getting stuck in local optima. In this way, pμ-CHC combines the evolutionary search with PALS in order to achieve high accurate results in short execution times. PALS is applied considering the best solutions found so far in the evolutionary search, which are stored in the elite population. In addition, a one-step memory is included: a task move is rejected when it will move the task back to the machine to which it was assigned one step in the past. The task memory is refreshed each time that a valid move is performed. This mechanism is a basic version of the memory already employed in TS algorithms to avoid loops, which has shown its usefulness for improving the HCSP results (Xhafa et al., 2008b).

Summarizing, the distinctive characteristics of pμ-CHC includes:

- Using a distributed subpopulation parallel model, with small populations within each deme (population size: eight individuals per deme).
- Storing a small elite population with the best three individuals found so far in the evolutionary search.
- Including a local search based on a randomized PALS method.
- Following an accelerated convergence model: the randomized PALS is applied after a certain (low) number of generations when a nominal convergence is detected.
- Using a one-step task memory to prevent loops in the task-to-machine assignments.

5.7.2 Parallel model of pμ-CHC

A two-level parallel model was used in the implementation of pμ-CHC applied to the HCSP: the distributed memory message-passing paradigm was employed for communicating demes that executes in different hosts in a distributed cluster, while the shared-memory paradigm was applied to improve the efficiency of the communications between demes executing in the same host. The hybrid parallel implementation of pμ-CHC allows taking advantage of two type of parallel infrastructures: traditional clusters of computers, and also modern multicore CPU architectures, where several processing cores share a global memory that can be used to speed up the communications in cooperative-based distributed search methods. Both communication paradigms were implemented using the Message Passing Interface (MPI) (Gropp et al., 1994), the most popular library used for developing parallel and distributed programs. By using the two-level parallel implementation, pμ-CHC diminishes the impact of the time spent in communication during the migration and synchronization procedures.

Figure 5.4 presents a graphical representation of the two-level parallel model used in pμ-CHC, showing a distributed PEA with many subpopulations arranged in a cluster with several multicore processors. The shared memory within each host is used for intra-processor communications, while a traditional message-passing migration operator is applied to communicate demes executing in different hosts.

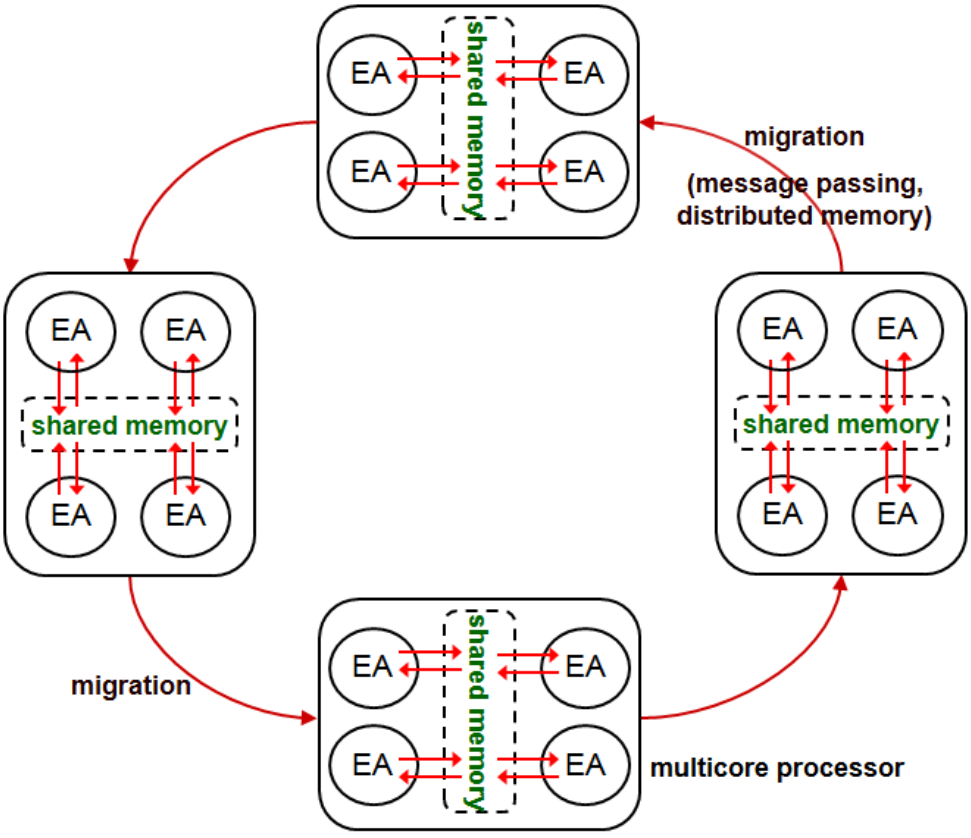


Figure 5.4: Two-level parallel model used in pμ-CHC.

5.8 Summary

This chapter presented the implementation details of the proposed EAs for the HCSP. MALLBA, the software library in which all the proposed algorithms were implemented, was introduced in the first section. The chapter also described the generic concepts about two alternative problem encoding and methods for the population initialization, which are common to all the proposed EAs.

The traditional GA uses SPX and a move-and-swap mutation operator, while CHC uses HUX, and an ad-hoc reinitialization operator specifically developed for the HCSP. However, when using these traditional methods for exploration and exploitation, the loss of diversity conspired against achieving accurate results for structured scenarios and large-sized problem instances. Thus, a new parallel micro-CHC evolutionary algorithm was proposed, inspired in previous methods for multimodal and multiobjective optimization.

The pμ-CHC algorithm uses HUX and the previously commented CHC reinitialization, but it also includes a local search performed by a randomized PALS method and an accelerated convergence model. A two-level parallel model was presented for the pμ-CHC implementation, which allows taking advantage of both the shared memory and the distributed memory high performance computing infrastructures.

The novel parallel micro-CHC algorithm was able to achieve high problem solving efficacy, even when facing large-sized HCSP instances. The details about the experimental analysis, results, and its discussion are provided in the next chapter.

Chapter 6

Experimental analysis

This chapter presents the experimental evaluation of serial and parallel EAs for solving the HCSP. The experimental analysis was aimed at studying the efficacy and efficiency of the proposed EAs for solving the low-dimension HCSP instances by Braun et al. (2001), and also analyzing the efficacy and scalability behavior when solving the new set of large-sized problem instances introduced in Section 3.4.

The first section briefly describes the set of HCSP instances used in the experimental evaluation (their details were already presented in Section 3.4). After that, additional tools used to develop the parallel EAs and the computational platform used for the experimental evaluation are presented. Next section presents and comments the experiments devoted to study the parameter settings of the proposed EAs. The last sections present and discuss the experimental results for sequential and parallel versions of the proposed algorithms when solving the problem instances considered. The report includes the numerical results, comparisons with other techniques and lower bounds, and statistical analysis on the results improvements. In addition, the analysis of the scalability and parallel performance of the parallel EAs is presented, and several studies on the behavior of the proposed EAs reporting interesting metrics (e.g. makespan evolution, time required to achieve certain levels of solution quality, etc.) are also included.

6.1 HCSP instances

The experimental evaluation was performed using several sets of HCSP instances. Regarding the low-sized HCSP instances, the de-facto standard benchmark instances from Braun et al. (2001) and a set of instances with 512 tasks and 16 machines designed using the ETC parametrization from Ali et al. (2000) were used. The scalability of the proposed algorithms for solving large-sized HCSP instances was studied using the HCSP instances specifically designed with dimension (tasks×machines) 1024×32, 2048×64, 4096×128, and 8192×256, whose details were presented in Section 3.4.3.

6.2 Development and execution platform

All the algorithms have been codified using the MALLBA library, implemented on C++. The parallel model employed in pμ-CHC used the version 1.2.7p1 of MPICH (Gropp et al., 1996), a well-known popular implementation of MPI, to perform the interprocess communications between demes. Both the distributed memory device (`ch_p4`) and the

shared memory device (`ch_shmem`) of MPICH were employed to implement the two-level parallel model proposed for pμ-CHC.

The experimental analysis was performed using a cluster of four Dell PowerEdge servers, with QuadCore Xeon E5430 processors at 2.66 GHz, 8 GB RAM, using the CentOS Linux 5.2 operating system, connected with a Gigabit Ethernet LAN at 1000 Mbps (cluster website: `http://www.fing.edu.uy/cluster`). This parallel computational infrastructure combines a shared memory architecture (the nodes within each processor) and a distributed memory architecture (the cluster itself). Thus, it allows applying the two-level parallel approach proposed for pμ-CHC: demes executing on the same processor use the shared memory to perform the inter-processes communications, while demes executing on different processors use the network to explicitly send and receive messages.

6.3 Parameter settings

Instead of fixing an arbitrary set of parameters, an initial configuration analysis was performed for determining the best parameter values for each operator used in the proposed EAs. The parameter setting experiments were performed using a subset of six problems with diverse characteristics from the set of instances by Braun et al. (`u_i_hilo.0`, `u_i_lohi.0`, `u_c_hihi`, `u_c_hilo.0`, `u_s_lohi.0`, and `u_s_lolo.0`). The studied parameters included: population size, crossover probability, mutation probability (in the GA), percentage of the population involved in the reinitialization (in CHC and pμ-CHC algorithms), and number of subpopulations (in the parallel algorithms). The operators probabilities and internal parameters were studied for all the EAs proposed. The analysis of the population size was performed for the sequential versions of GA and CHC, while the analysis of the number of subpopulations was performed for the parallel CHC and pμ-CHC algorithms. For each parameter configuration analysis, 30 executions of the correspondent algorithm were performed for each problem instance and each parameter configuration studied.

This section presents the numerical analysis and comments the main decisions taken and the experiments performed to study the parameter settings of the proposed EAs.

6.3.1 Stopping criterion

The main objective of the research is to study the ability of the proposed EAs to *efficiently* solve the HCSP, thus demonstrating their usefulness to act as practical schedulers for real-life sized HC and grid systems. Thus, the algorithms studied in this work used a bounded time stopping criterion, following previous works by Carretero and Xhafa (2006) and Xhafa et al. (2007b, 2008a,b), in order to determine the ability of EAs to achieve accurate HSCP solutions in short execution times. This decision is useful for efficiently solving static HCSP instances, and is also useful for solving dynamic scenarios following the rescheduling strategy, by replanning incoming –and also unexecuted– tasks after certain intervals of time.

When dealing with small and medium-sized HCSP instances, the bounded effort stopping criterion was fixed at 90 s. of wall-clock time (following related works by Xhafa et al.), while when solving the large HCSP instances the stopping criterion was fixed at 120 s. of execution time. These time limits can be considered too long for scheduling short tasks in small multiprocessors, but it is actually an efficient time for scheduling in

realistic distributed HC and grid infrastructures such as volunteer-computing platforms, distributed databases, etc., where large tasks –with execution times in the order of minutes, hours and even days– are submitted to execution. In addition, when facing large HCSP instances, ad-hoc deterministic heuristics that usually require $O(n^3)$ operations also require execution times in the order of minutes (e.g. Min-Min needs around a minute of execution time for computing schedules for HCSP instances with dimension 8192×256). The stopping criterion used in the parallel EAs is useful for efficiently solving static HCSP instances, and is also useful for solving dynamic scenarios following the rescheduling strategy, by replanning incoming and unexecuted tasks after certain intervals of time.

6.3.2 Population size

The population size configuration analysis was performed for the sequential versions of GA and CHC. The candidate values for population size were 60, 120, and 200 individuals. The best makespan values were achieved when using a population size of 120 individuals. The analysis showed a very slow makespan evolution behavior for the sequential algorithms when increasing the population size from 120 to 200, suggesting that working with a larger population is not beneficial for the EAs: the processing time increases, thus conspiring against quickly achieving accurate results.

6.3.3 Operators probabilities

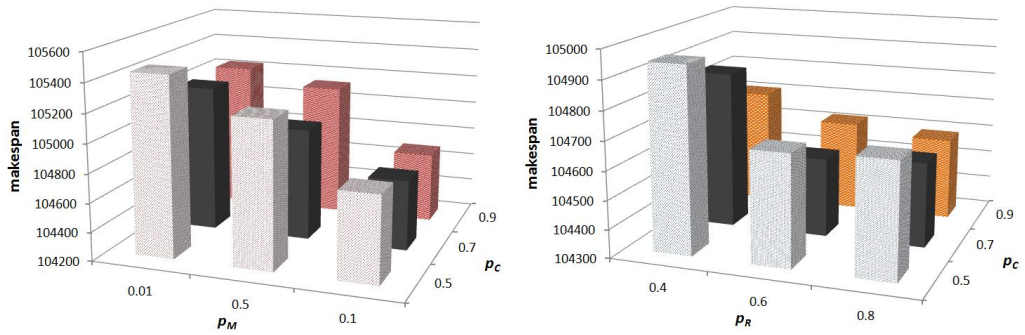
The candidate values for the application probabilities of the evolutionary operators were:

- Crossover probability (p_C), in GA and CHC: 0.5, 0.7, 0.9.
- Mutation probability, in GA (p_M): 0.01, 0.05, 0.1.
- Percentage of reinitialization, in CHC and pμ-CHC (p_R): 0.4, 0.6, 0.8.

The best makespan results were achieved when using the following operators probabilities configurations:

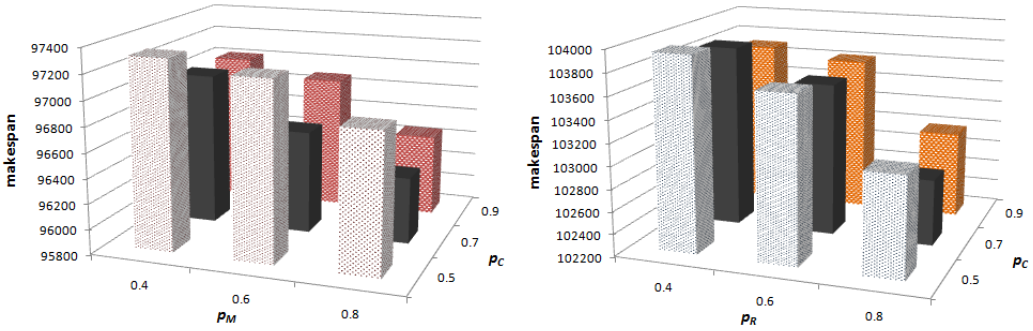
- **GA** (population size=120): $p_C = 0.7$, $p_M = 0.1$.
- **CHC** (population size=120): $p_C = 0.7$, $p_R = 0.6$.
- **parallel CHC** (population size=15): $p_C = 0.7$, $p_R = 0.8$.
- **pμ-CHC** (population size=8): $p_C = 0.7$, $p_R = 0.8$.

Figures 6.1 and 6.2 present examples of the results obtained in representative executions of the configuration setting experiments for the sequential and parallel algorithms, respectively. The graphics report the average makespan values achieved for each combination of p_C and p_M (or p_R) in 30 executions of the correspondent EA for solving the problem instances `u_i_hilo.0` and `u_s_hilo.0`, when using the stopping criterion of 90 s. Figure 6.1 presents the sample results for sequential GA and CHC, using a population size of 120 120 individuals, and Figure 6.2 for pCHC –splitting the panmictic population in 8 demes (15 individuals per deme)– and pμ-CHC –using 16 subpopulations with 8 individuals inside each deme–. Similar results were obtained for the other problem instances studied.



(a) GA (b) CHC

Figure 6.1: Analysis of the operators probabilities (sequential algorithms, `u_i_hilo.0`).



(a) parallel CHC, `u_s_hilo.0` (b) pμ-CHC `u_i_hilo.0`

Figure 6.2: Analysis of the operators probabilities (parallel algorithms).

The results obtained in the operators probabilities configuration experiments demonstrate the importance of the mutation and reinitialization operators to achieve highly accurate solutions in short execution times: the best results are obtained when using (unusual) high mutation probability (in GA) and percentage of reinitialization (in CHC). These results are consistent with related works that have claimed that specific methods are needed to provide diversity in order to achieve accurate HCSP solutions in short execution times (Xhafa et al., 2007b, 2008b).

6.3.4 Parallel algorithms

The parallel EAs use a migration operator that exchanges individuals considering the subpopulations connected in an unidirectional ring topology. The best results were achieved when employing an elitist selection for migration policy that exchanges the best two individuals between demes. The parameter setting analysis showed no significant makespan variations when changing the migration frequency, so the algorithms worked using a migration frequency of 500 generations trying to achieve a balance between providing diversity and reducing the time spent in communications.

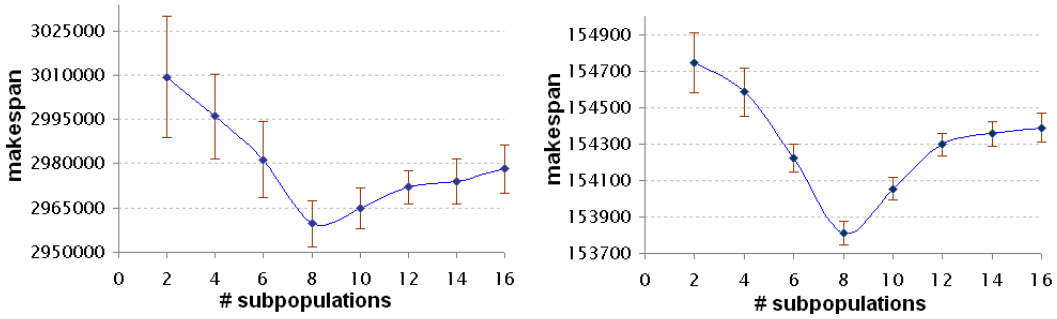
Regarding the number of subpopulations, the parallel EAs showed a different behavior. In the parallel versions of the traditional GA and CHC algorithms, the best results were achieved when using eight subpopulations, while pμ-CHC was able to obtain the best results when using the largest number of subpopulations considered (16). The details of the configuration analysis for the parallel versions of CHC are provided in the next subsections.

Number of subpopulations: parallel CHC

Table 6.1 and Figure 6.3 present representative results achieved in the analysis that studied the ability of the pCHC algorithm of obtaining improved results when working with additional subpopulations. A population of 120 individuals was split into 2 to 16 subpopulations and the parallel CHC algorithm was run with the stopping criteria of 90 s. for solving the HCSP instances `u_i_hihi.0`, `u_c_hilo.0`, and `u_s_hilo.0`. The average and standard deviation of the makespan results are reported for each problem instance studied.

Instance		# subpopulations							
		2	4	6	8	10	12	14	16
`u_i_hihi.0`	avg.	3009246.7	2995968.1	2981391.5	**2959721.8**	2964912.4	2972011.4	2974025.4	2978223.4
	σ	0.68%	0.48%	0.43%	0.26%	0.23%	0.19%	0.25%	0.28%
`u_c_hilo.0`	avg.	154745.1	154585.4	154223.1	**153811.5**	154055.2	154298.5	154356.5	154391.0
	σ	0.11%	0.08%	0.05%	0.04%	0.04%	0.04%	0.05%	0.05%
`u_s_lohi.0`	avg.	124886.2	124603.6	124329.8	**124204.4**	124523.6	124442.0	124581.8	124852.4
	σ	0.43%	0.57%	0.46%	0.55%	0.42%	0.38%	0.30%	0.26%

Table 6.1: Sample results when configuring the number of subpopulations in pCHC.



(a) `u_i_hihi.0` (b) `u_c_hilo.0`.

Figure 6.3: Sample results when configuring the number of subpopulations in pCHC.

When using smaller populations, pCHC has a more focused exploration pattern, so it is able to take advantage of the evolutionary search in an efficient manner (see the decrease on average makespan values when using more than one subpopulation in Figure 6.3). However, splitting the population in more than 8 demes causes a severe lost of diversity and the parallel CHC results deteriorated (the same drawback was detected for the parallel GA algorithm).

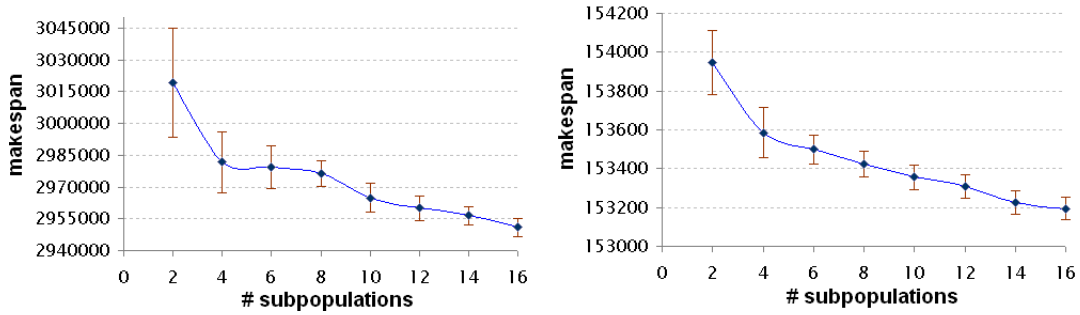
The previous scalability behavior suggested that there was still work to be done in order to enhance the pCHC method, and that was one of the main reasons for designing the improved parallel micro-CHC algorithm.

Number of subpopulations: pμ-CHC

Table 6.2 and Figure 6.4 present representative results achieved in an analysis that studied the ability of pμ-CHC of finding more accurate results when working with additional subpopulations. As in the previous study, a population of 120 individuals was split into 2 to 16 subpopulations and the pμ-CHC algorithm was run with the stopping criteria of 90 s. for solving the HCSP instances `u_i_hihi.0`, `u_c_hilo.0`, and `u_s_hilo.0`. The average and standard deviation of the makespan results are reported for each problem instance studied.

Instance		# subpopulations							
		2	4	6	8	10	12	14	16
u_i_hihi.0	avg.	124886.2	124603.6	124329.8	124204.4	123976.8	123442.0	123081.8	**122852.4**
	σ	0.43%	0.57%	0.46%	0.55%	0.42%	0.38%	0.31%	0.27%
u_c_hilo.0	avg.	3019247	2981653	2981392	2976351	2964912	2960011	2956506	**2950896.4**
	σ	0.85%	0.48%	0.43%	0.20%	0.23%	0.19%	0.15%	0.14%
u_s_hilo.0	avg.	153945.1	153585.4	153498.5	153423.1	153355.2	153307.8	153226.4	**153193.7**
	σ	0.11%	0.08%	0.05%	0.04%	0.04%	0.04%	0.04%	0.04%

Table 6.2: Sample results when configuring the number of subpopulations in pμ-CHC.



(a) `u_i_hihi.0` (b) `u_c_hilo.0`.

Figure 6.4: Sample results when configuring the number of subpopulations in pμ-CHC.

When using smaller populations, pμ-CHC has a more focused exploration pattern, so it is able to take advantage of both the evolutionary search and the randomized PALS operator in an efficient way. The analysis also demonstrates that pμ-CHC overcomes the problem previously detected in the pCHC algorithm: the diversity provided by the external population and the randomized PALS operator allows pμ-CHC to further improve the makespan results when using additional demes (using additional computational resources), despite working with smaller subpopulations.

6.3.5 Probabilities within the evolutionary operators

Additional experiments were performed in order to find the optimum values for the probabilities used inside the mutation, reinitialization and PALS operators. After performing the configuration analysis, the value of HEAVY_MACH (the probability of selecting the machine with the largest local makespan) was fixed to 0.7. The value of MAX_TRIALS (number of attempts to find a better solution before accepting a worse one) was set to seven in the GA mutation, and to five in the CHC and pμ-CHC reinitialization.

Regarding the randomized PALS method, the best results were achieved applying it after five generations pass without inserting any offspring into the new population during the mating procedure. The value of H_M (the probability of selecting the machine with the largest local makespan) was fixed to 0.7. The value of TOP_M (number of tasks of the heavy machine to process) was fixed to N/M (32 for all the problem instances studied), while the value of TOP_T (number of tasks of other machines examined to swap) was fixed to $N/20$. The number of iterations attempts to find a solution with lower makespan (MAXIT_PALS) was set to 7.

The previously presented parameter values allow an efficient search of the HCSP solution space: only TOP_M×TOP_T swaps are explored in every application of PALS. Since TOP_M×TOP_T $= N/M \times N/20$ and $N = M * 32$ for all problem instances studied, the number of swaps explored is $32 \times 32 * M/5 \cong 50 * M$ –linear order on both the number of tasks (N) and machines (M)–.

6.3.6 Summary: best parameter configurations

In summary, all the studied EAs worked using a bounded time stopping criteria of 90 s. for the HCSP instances by Braun et al. (2001). The best results in the configuration analysis were obtained when using the parameters values presented in Tables 6.3 (sequential algorithms) and 6.4 (parallel algorithms).

	pop. size	*exploitation*	*exploration*	*additional parameters*
GA	120	p_C=0.7	p_M=0.1	H_M=0.7, MAX_TRIALS=7
CHC	120	p_C=0.7	p_R=0.6	H_M=0.7, MAX_TRIALS=5

Table 6.3: Best parameter configurations (sequential algorithms).

	#demes	*pop. size*	*exploitation*	*exploration*	*additional parameters*
pCHC	8	15	p_C=0.7	p_R=0.8	MAX_TRIALS=5, H_M=0.7
pμ-CHC	16	8	p_C=0.7	p_R=0.8	TOP_M=N/M,TOP_T=$N/20$, MAXIT_PALS=7

Table 6.4: Best parameter configurations (parallel algorithms).

The parallel algorithms used an elitist selection for migration, a replace-worst replacement strategy, and a migration frequency of 500 generations. In pμ-CHC, the randomized PALS method was applied after five generations pass without inserting any offspring into the new population during the mating procedure.

6.4 Results and discussion

This section presents the experimental results obtained with the proposed EAs for solving the HCSP. It studies the numerical results for the set of HCSP instances from Braun et al. (2001), a statistical analysis on the improvements obtained with the parallel EAs, and the comparison with other metaheuristics. After that, the results obtained with the parallel EAs when solving the large-dimension HCSP instances specially designed in this work are presented and discussed. Finally, the section includes the analysis of the makespan evolution, a comparison with lower bounds, and a study of the parallel performance and scalability of the parallel EAs when solving large-sized HCSP instances.

6.4.1 Results for HCSP instances from Braun et al. (2001)

The results for the set of HCSP instances from Braun et al. (2001) (dimension 512×16) are presented separately, since there have been antecedents of solving this benchmark using other methods. Considerably effort has been done in this work in order to design accurate algorithms for the HCSP, able to improve over the previous results using EAs and other metaheuristics for the problem instances studied. Next subsections present the numerical results using sequential and parallel EAs.

Sequential algorithms

The results achieved using the sequential versions of GA and CHC for the instances from Braun et al. (2001) are presented in Table 6.5. The table shows the best, average and standard deviation (σ) of the makespan values obtained in 50 independent executions of the sequential GA and CHC for solving each of the twelve problem instances studied. The best makespan results obtained for each instance are marked with bold face.

Instance	GA			CHC		
	best	avg.	σ	best	avg.	σ
u_c_hihi.0	7659878.7	7699080.1	0.41%	**7599288.4**	7681050.1	0.55%
u_c_hilo.0	155092.0	155300.1	0.11%	**154947.0**	155333.4	0.19%
u_c_lohi.0	**250511.8**	252568.7	0.56%	251194.3	251868.3	0.22%
u_c_lolo.0	5239.1	5248.6	0.16%	**5225.9**	5241.9	0.20%
u_i_hihi.0	3019844.3	3030564.2	0.22%	**3015048.5**	3024904.9	0.30%
u_i_hilo.0	**74142.9**	74568.4	0.41%	74240.9	74375.9	0.12%
u_i_lohi.0	104688.0	105048.1	0.31%	**104546.0**	104939.1	0.32%
u_i_lolo.0	2577.0	2587.8	0.46%	**2576.7**	2582.2	0.13%
u_s_hihi.0	4332248.2	4347835.5	0.38%	**4299146.3**	4320803.4	0.52%
u_s_hilo.0	**97630.1**	98026.1	0.41%	97888.2	98307.4	0.27%
u_s_lohi.0	126438.0	126840.8	0.25%	**126238.0**	126580.4	0.20%
u_s_lolo.0	3510.4	3516.5	0.23%	**3492.1**	3505.0	0.25%

Table 6.5: Results of the sequential EAs for the HCSP.

The analysis of Table 6.5 indicates that CHC systematically achieved better results than GA for nine out of twelve problem instances studied (GA only obtained slightly better makespan values than CHC for u_c_lohi.0, u_i_hilo.0, and u_s_hilo.0). These results suggested that the CHC evolutionary model –using HUX, diversity preservation, and population reinitialization–, is a promising strategy for solving the HCSP when compared with a traditional GA model.

Instance	Min-Min	Sufferage	GA	imp.	CHC	imp.
u_c_hihi.0	8460674.0	10249174.0	7659878.7	9.46%	7599288.4	**10.18%**
u_c_hilo.0	161805.4	168982.6	155092	4.15%	154947	**4.24%**
u_c_lohi.0	275837.3	337121.4	250511.8	**9.18%**	251194.3	8.93%
u_c_lolo.0	5441.4	5658.5	5239.1	3.72%	5225.9	**3.96%**
u_i_hihi.0	3513919.3	3306819.3	3019844.3	8.68%	3015048.5	**8.82%**
u_i_hilo.0	80755.7	77589.1	74142.9	**4.44%**	74240.9	4.32%
u_i_lohi.0	120517.7	114578.9	104688	8.63%	104546	**8.76%**
u_i_lolo.0	2785.6	2639.3	2577	2.36%	2576.7	**2.37%**
u_s_hihi.0	5160343.0	5121953.5	4332248.2	15.42%	4299146.3	**16.06%**
u_s_hilo.0	104375.2	102499.9	97630.1	**4.75%**	97888.2	4.50%
u_s_lohi.0	140284.5	150297.1	126438	9.87%	126238	**10.01%**
u_s_lolo.0	3806.8	3846.5	3510.4	7.79%	3492.1	**8.27%**

Table 6.6: Comparative results: sequential EAs vs. deterministic heuristics.

Table 6.6 presents a comparative analysis between the best results obtained with the sequential EAs and those computed by Min-Min and Sufferage, the deterministic heuristics used to define the randomized procedure for the population initialization. The results show that both GA and CHC are able to achieve significant improvement factors (imp.) over the best makespan result computed using a deterministic heuristic: the improvement factors ranged from 2.36% to 15.42% (for GA) and from 2.37% to 16.06% (for CHC). The overall improvement factors –averaged for the twelve problem instances studied– were **7.37%** for GA and **7.54%** for CHC.

Table 6.7 compares the best makespan values achieved by the sequential GA and CHC against the previous best results obtained with EAs for the HCSP. The analysis of Table 6.7 shows that both sequential EAs proposed in this work outperformed previous results achieved with evolutionary techniques for six (four inconsistent and two semi-consistent) out of twelve problem instances studied. However, the sequential methods were not able to achieve the best makespan results formerly known for each of the benchmark HCSP instances, achieved with non-evolutionary methods by Ritchie and Levine (2004) and Xhafa et al. (2008b). The best results obtained with the sequential EAs designed in this work were around 1% far from the best-known makespan results.

Instance	GA (Braun et al.)	GA (Xhafa)	MA+TS (Xhafa et al.)	cMA (Xhafa et al.)	GA (this work)	CHC (this work)
u_c_hihi.0	8050844.5	7610176.7	**7530020.2**	7700929.8	7659879	7599288.4
u_c_hilo.0	156249.2	155251.2	**153917.2**	155334.8	155092	154947.0
u_c_lohi.0	258756.8	248466.8	**245288.9**	251360.2	250511.8	251194.3
u_c_lolo.0	5272.3	5227.0	**5173.7**	5218.2	5239.1	5225.9
u_i_hihi.0	3104762.5	3077705.8	3058474.9	3186664.7	3019844	**3015048.5**
u_i_hilo.0	75816.1	75924.0	75108.5	75856.6	**74142.9**	74240.9
u_i_lohi.0	107500.7	106069.1	105808.6	110620.8	104688	**104546.0**
u_i_lolo.0	2614.4	2613.1	2596.6	2624.2	2577	**2576.7**
u_s_hihi.0	4566206.0	4359312.6	4321015.4	4424540.9	4332248	**4299146.3**
u_s_hilo.0	98519.4	98334.6	**97177.3**	98283.7	97630.1	97888.2
u_s_lohi.0	130616.5	127641.9	127633.0	130014.5	126438	**126238.0**
u_s_lolo.0	3583.4	3515.5	**3484.1**	3522.1	3510.4	3492.1

Table 6.7: Comparative results: sequential EAs vs. previous EAs for the HCSP.

Parallel GA and parallel CHC

The results achieved using the parallel versions of GA (pGA) and CHC (pCHC) for the set of instances from Braun et al. are presented in Table 6.8. The table shows the best, average and standard deviation (σ) of the makespan values computed in 50 independent executions of pGA and pCHC for solving each of the twelve problem instances studied.

Instance	parallel GA			parallel CHC		
	best	avg.	σ	best	avg.	σ
u_c_hihi.0	7577921.9	7606613.0	0.29%	**7461819.1**	7481194.5	0.26%
u_c_hilo.0	154915.0	155036.5	0.06%	**153791.9**	153924.0	0.06%
u_c_lohi.0	248772.4	249687.9	0.33%	**241513.2**	243446.3	0.29%
u_c_lolo.0	5208.3	5224.7	0.17%	**5177.5**	5181.6	0.07%
u_i_hihi.0	2990517.8	3002119.3	0.25%	**2952493.2**	2956905.7	0.21%
u_i_hilo.0	74030.3	74102.8	0.21%	**73639.8**	73847.1	0.13%
u_i_lohi.0	103516.0	104078.6	0.34%	**102123.1**	102677.3	0.30%
u_i_lolo.0	2575.4	2577.0	0.12%	**2548.9**	2557.2	0.11%
u_s_hihi.0	4262337.5	4282920.5	0.25%	**4198779.5**	4239146.3	0.36%
u_s_hilo.0	97505.5	97585.5	0.05%	**96623.3**	96750.3	0.13%
u_s_lohi.0	125717.0	126100.1	0.22%	**123236.9**	123989.4	0.24%
u_s_lolo.0	3480.3	3487.2	0.11%	**3450.1**	3472.2	0.13%

Table 6.8: Results of parallel EAs for the HCSP.

The analysis of Table 6.8 shows that pCHC consistently achieved best results than pGA for the twelve HCSP instances studied. The comparison of Tables 6.5 and 6.8 demonstrates that the parallel models of EAs are able to significantly improve over the results of the sequential algorithms (achieving an improvement factor of up to 4% for u_c_lohi.0). The parallel algorithms take advantage of the multiple search pattern and the increased diversity provided by the subpopulation model to improve the evolutionary search. In addition, since they work with a reduced population, the parallel EAs have a more focused processing capability, which allows them to achieve highly accurate results when using the bounded execution time stopping criterion of 90 s. The standard deviation of the makespan values are very small for the two parallel EAs –well below 0.5%–, demonstrating a high robustness behavior when solving the HCSP.

Instance	GA		CHC		parallel GA	
	imp.	p-value	imp.	p-value	imp.	p-value
u_c_hihi.0	2.65%	$< 10^{-4}$	1.84%	$< 10^{-4}$	1.56%	$< 10^{-4}$
u_c_hilo.0	0.85%	1×10^{-4}	0.75%	6×10^{-3}	0.73%	5×10^{-3}
u_c_lohi.0	3.72%	$< 10^{-4}$	4.00%	$< 10^{-4}$	3.00%	$< 10^{-4}$
u_c_lolo.0	1.19%	$< 10^{-4}$	0.93%	1×10^{-4}	0.59%	3×10^{-3}
u_i_hihi.0	2.28%	$< 10^{-4}$	2.12%	$< 10^{-4}$	1.29%	1×10^{-4}
u_i_hilo.0	0.68%	7×10^{-3}	0.82%	1×10^{-4}	0.53%	4×10^{-3}
u_i_lohi.0	2.50%	$< 10^{-4}$	2.36%	$< 10^{-4}$	1.35%	$< 10^{-4}$
u_i_lolo.0	1.07%	$< 10^{-4}$	1.05%	$< 10^{-4}$	1.00%	$< 10^{-4}$
u_s_hihi.0	3.18%	$< 10^{-4}$	2.39%	$< 10^{-4}$	1.51%	$< 10^{-4}$
u_s_hilo.0	1.04%	$< 10^{-4}$	1.31%	$< 10^{-4}$	0.91%	2×10^{-4}
u_s_lohi.0	2.59%	$< 10^{-4}$	2.42%	$< 10^{-4}$	2.00%	$< 10^{-4}$
u_s_lolo.0	1.75%	$< 10^{-4}$	1.22%	$< 10^{-4}$	0.88%	2×10^{-4}

Table 6.9: Improvement factors and statistical analysis when using parallel CHC.

Table 6.9 shows the improvement factors obtained when using pCHC over the other studied methods. The Kruskal-Wallis test was performed to analyze the time distributions, and the correspondent p-values are presented for each problem instance and each pairwise algorithm comparison. The pCHC makespan values significantly improve over the results achieved with the other EAs, and the computed p-values are very small, so the improvements can be considered as statistically significant.

Comparison: parallel CHC against other methods

The parallel versions of GA and CHC achieved lower makespan values with respect to their sequential counterparts, but pGA was unable to improve over the best makespan results formerly known for the problem instances from Braun et al. (2001). The pCHC algorithm achieved more accurate schedules indeed, as it is discussed below.

Table 6.10 compares the best results obtained with pCHC for the instances from Braun et al. (2001) against the best results previously found with diverse metaheuristic techniques. The comparison with ACO+TS (Ritchie and Levine, 2004) and TS (Xhafa et al., 2008b) is specially relevant, since those methods have achieved the previous best-known results to date for the HCSP instances studied. In those cases where pCHC outperformed the previous best results, the mean time required to achieve the previous best-known makespan value (t_B, in s.) is also presented.

Instance	GA (Braun et al.)	MA+TS (Xhafa)	cMA (Xhafa et al.)	ACO+TS (Ritchie et al.)	TS (Xhafa et al.)	pCHC best	pCHC avg.	pCHC σ	pCHC t_B
u_c_hihi.0	8050844.5	7530020.2	7700929.8	7497200.9	**7448640.5**	7461819.1	7481194.5	0.26%	-
u_c_hilo.0	156249.2	153917.2	155334.8	154234.6	**153263.3**	153791.9	153924.0	0.06%	-
u_c_lohi.0	258756.8	245288.9	251360.2	244097.3	241672.7	**241524.0**	243446.3	0.29%	71
u_c_lolo.0	5272.3	5173.7	5218.2	5178.4	**5155.0**	5177.5	5181.6	0.07%	-
u_i_hihi.0	3104762.5	3058474.9	3186664.7	**2947754.1**	2957854.1	2952493.2	2956905.7	0.21%	-
u_i_hilo.0	75816.1	75108.5	75856.6	73776.2	73692.9	**73639.8**	73847.1	0.13%	74
u_i_lohi.0	107500.7	105808.6	110620.8	102445.8	103865.7	**102136.1**	102677.3	0.30%	31
u_i_lolo.0	2614.4	2596.6	2624.2	2553.5	2552.1	**2549.8**	2557.2	0.11%	62
u_s_hihi.0	4566206	4321015.4	4424540.9	**4162547.9**	4168795.9	4198779.5	4239146.3	0.36%	-
u_s_hilo.0	98519.4	97177.3	98283.7	96762	**96180.9**	96623.3	96750.3	0.13%	-
u_s_lohi.0	130616.5	127633	130014.5	123922	123407.4	**123251.5**	123989.4	0.24%	55
u_s_lolo.0	3583.4	3484.1	3522.1	3455.2	3450.5	**3450.1**	3472.2	0.13%	80

Table 6.10: pCHC and other methods for HCSP instances by Braun et al. (2001).

The analysis of Table 6.10 shows that pCHC outperformed the results obtained with previous EAs. pCHC also outperformed the ACO+TS by Ritchie and Levine (2004) in ten out of twelve HCSP instances, and the TS by Xhafa et al. (2008b) in seven out of twelve HCSP instances. In addition, pCHC was able to compute schedules with better makespan values than the previously best-known solutions in six problem instances (the correspondent makespan values are marked with bold in Table 6.10). Short execution times are required to outperform the previous results in those cases.

Figure 6.5 summarizes the information about whether pCHC improves over the previous best-known methods for the problem set by Braun et al. (2001) or not, and it also states when the pCHC algorithm obtained a best-known solution for each HCSP instance studied. The solutions (schedules) with the lowest makespan values obtained for each problem instance are reported in Appendix B and further details are provided at the HCSP website (`http://www.fing.edu.uy/inco/cecal/hpc/HCSP`).

Instance	parallel CHC		
	is better than		achieves a best known solution
	ACO+TS (Ritchie,Levine)	TS (Xhafa et al.)	
u_c_hihi.0	YES	NO	NO
u_c_hilo.0	YES	NO	NO
u_c_lohi.0	YES	YES	YES
u_c_lolo.0	YES	NO	NO
u_i_hihi.0	NO	YES	NO
u_i_hilo.0	YES	YES	YES
u_i_lohi.0	YES	YES	YES
u_i_lolo.0	YES	YES	YES
u_s_hihi.0	NO	NO	NO
u_s_hilo.0	YES	NO	NO
u_s_lohi.0	YES	YES	YES
u_s_lolo.0	YES	YES	YES
TOTAL	10 out of 12	7 out of 12	6 out of 12

Figure 6.5: Comparison of pCHC against the best-known methods for the HCSP.

The previous results show that the parallel version of the CHC algorithm is a promising method for solving the set of low-dimension HCSP instances by Braun et al. (2001).

Parallel micro-CHC

Table 6.11 reports the best, average and standard deviation of the makespan results obtained in 50 independent executions of pμ-CHC for the set of instances by Braun et al. For the sake of comparison, Table 6.11 also reports the values obtained with the Min-Min and Sufferage deterministic heuristics used to define the randomized procedure for the population initialization and the pCHC method, reported in the previous subsection.

Instance	Min-Min	Sufferage	pCHC			pμ-CHC		
			best	avg.	σ	best	avg.	σ
u_c_hihi.0	8460674.0	10249174.0	7461819	7481195	0.26%	**7381570.0**	**7394702.7**	**0.09%**
u_c_hilo.0	161805.4	168982.6	153791.9	153924	0.06%	**153105.4**	**153193.7**	**0.04%**
u_c_lohi.0	275837.3	337121.4	241524	243446.3	0.29%	**239260.0**	**239706.2**	**0.08%**
u_c_lolo.0	5441.4	5658.5	5177.5	5181.6	0.07%	**5147.9**	**5152.3**	**0.04%**
u_i_hihi.0	3513919.3	3306819.3	2952493	2956906	0.21%	**2938380.8**	**2947896.4**	**0.14%**
u_i_hilo.0	80755.7	77589.1	73639.8	73847.1	0.13%	**73378.0**	**73531.4**	**0.10%**
u_i_lohi.0	120517.7	114578.9	102136.1	102677.3	0.30%	**102050.6**	**102402.8**	**0.17%**
u_i_lolo.0	2785.6	2639.3	2549.8	2557.2	0.11%	**2541.4**	**2547.1**	**0.09%**
u_s_hihi.0	5160343.0	5121953.5	4198780	4239146	0.36%	**4103500.3**	**4123537.3**	**0.27%**
u_s_hilo.0	104375.2	102499.9	96623.3	96750.3	0.13%	**95787.4**	**96020.5**	**0.10%**
u_s_lohi.0	140284.5	150297.1	123251.5	123989.4	0.24%	**122083.3**	**122744.4**	**0.23%**
u_s_lolo.0	3806.8	3846.5	3450.1	3472.2	0.13%	**3433.5**	**3438.3**	**0.07%**

Table 6.11: Results of pμ-CHC for HCSP instances from Braun et al. (2001).

The results in Table 6.11 show that pμ-CHC obtained significant improvements over the best deterministic heuristic result: the improvement factors were from 3.71% to 19.88%, with an averaged **9.77%** overall improvement factor. Table 6.11 also shows that the new algorithmic proposal in pμ-CHC improves over all the results previously obtained with the traditional pCHC algorithm. By using a micro population, the accelerated convergence model, the elite population, and the local search based on a randomized PALS method in order to improve the population diversity, pμ-CHC is able to find accurate HCSP solutions with lower makespan values than the previously obtained with pCHC. The algorithmic robustness was also improved in pμ-CHC, since very reduced values for the standard deviation of the makespan results were obtained. Thus, it can be expected that pμ-CHC will find accurate schedules in any single execution for small-sized HCSP scenarios that follow the ETC performance estimation model by Ali et al. (2000).

Table 6.12 presents the improvement factors on the makespan values obtained with pμ-CHC over pCHC. The p-values from the non-parametric Kruskal-Wallis test, performed to analyze the distributions, are reported for each problem instance. The pμ-CHC makespan values improve over the pCHC results, and the computed p-values are very small, so the improvements can be considered as statistically significative.

Instance	**avg**		**best**	
	imp.	p-value	imp.	p-value
u_c_hihi.0	1.16%	$< 10^{-3}$	1.08%	$< 10^{-3}$
u_c_hilo.0	0.47%	$< 10^{-3}$	0.45%	0.009
u_c_lohi.0	1.54%	$< 10^{-3}$	0.93%	$< 10^{-3}$
u_c_lolo.0	0.57%	$< 10^{-3}$	0.57%	$< 10^{-3}$
u_i_hihi.0	0.30%	0.005	0.48%	0.003
u_i_hilo.0	0.43%	$< 10^{-3}$	0.36%	0.009
u_i_lohi.0	0.27%	0.005	0.07%	0.01
u_i_lolo.0	0.39%	$< 10^{-3}$	0.29%	$< 10^{-3}$
u_s_hihi.0	2.73%	$< 10^{-3}$	2.27%	$< 10^{-3}$
u_s_hilo.0	0.75%	$< 10^{-3}$	0.87%	$< 10^{-3}$
u_s_lohi.0	1.00%	$< 10^{-3}$	0.94%	$< 10^{-3}$
u_s_lolo.0	0.98%	$< 10^{-3}$	0.48%	$< 10^{-3}$

Table 6.12: pμ-CHC improvement factors over pCHC.

Table 6.13 presents the comparison of the pμ-CHC best makespan values against the best results previously found with other EAs and diverse metaheuristic techniques. Table 6.13 also presents the mean time required by pμ-CHC to compute the previous best-known makespan value (t_B, in seconds). The analysis of Table 6.13 shows that pμ-CHC was able to compute better makespan values than the previously best-known solutions for all problem instances in the de-facto set of benchmark problems by Braun et al. (2001). pμ-CHC outperformed the ACO+TS by Ritchie and Levine (2004), the TS by Xhafa et al. (2008b), and pCHC, the three methods that have obtained the previous best-known solutions for the HCSP instances by Braun et al. Short execution times are required to outperform the previous best-known results in all cases. The makespan values for the best solutions obtained are marked with bold in Table 6.13). The best solutions (schedules) obtained for each problem instance are presented in Appendix B and they are also reported in the HCSP website `http://www.fing.edu.uy/inco/cecal/hpc/HCSP`.

Instance	GA (Braun et al.)	MA+TS (Xhafa et al.)	cMA (Xhafa et al.)	ACO+TS (Ritchie et al.)	TS (Xhafa et al.)	pCHC	pμ-CHC	t_B (s.)
u_c_hihi.0	8050844.5	7530020.2	7700929.8	7497200.9	7448640.5	7461819.1	**7381570.0**	15
u_c_hilo.0	156249.2	153917.2	155334.8	154234.6	153263.3	153791.9	**153105.4**	62
u_c_lohi.0	258756.8	245288.9	251360.2	244097.3	241672.7	241524.0	**239260.0**	23
u_c_lolo.0	5272.3	5173.7	5218.2	5178.4	5155.0	5177.5	**5147.9**	50
u_i_hihi.0	3104762.5	3058474.9	3186664.7	2947754.1	2957854.1	2952493.2	**2938380.8**	44
u_i_hilo.0	75816.1	75108.5	75856.6	73776.2	73692.9	73639.8	**73378.0**	43
u_i_lohi.0	107500.7	105808.6	110620.8	102445.8	103865.7	102136.1	**102050.6**	57
u_i_lolo.0	2614.4	2596.6	2624.2	2553.5	2552.1	2549.8	**2541.4**	49
u_s_hihi.0	4566206	4321015.4	4424540.9	4162547.9	4168795.9	4198779.5	**4103500.3**	18
u_s_hilo.0	98519.4	97177.3	98283.7	96762	96180.9	96623.3	**95787.4**	46
u_s_lohi.0	130616.5	127633	130014.5	123922	123407.4	123251.5	**122083.3**	24
u_s_lolo.0	3583.4	3484.1	3522.1	3455.2	3450.5	3450.1	**3433.5**	30

Table 6.13: pμ-CHC and other methods for HCSP instances by Braun et al. (2001).

After the previously commented results, we can claim that pμ-CHC is the new state-of-the-art algorithm for solving the low-dimension HCSP instances by Braun et al. Next subsection characterizes the performance of the proposed parallel EAs for solving unseen larger problems.

6.5 Results for the new large-sized HCSP instances

This section presents and discusses the results obtained when solving the new large dimension HSCP instances specially designed in this work, using the parallel versions of CHC and pμ-CHC (identified as the best methods for solving the HCSP in the previous section). For each problem dimension (1024×32, 2048×64, 4096×128, and 8192×256 tasks×machines), the results for the twelve instances following the heterogeneity model from Ali et al. (2000), and the twelve ones following the heterogeneity model from Braun et al. (2001) are presented in the next subsections. There have been no previous works solving this new set of HCSP instances, so the results obtained using the parallel EAs are compared with those achieved by the popular ad-hoc deterministic list scheduling heuristics used to define the randomized procedure for the population initialization in the proposed EAs (MCT, Min-Min, and Sufferage).

6.5.1 Parallel CHC

Tables 6.14 to 6.17 present the results obtained when using pCHC for solving the large HSCP instances with dimension 1024×32, 2048×64, 4096×128, and 8192×256, respectively. The results obtained with the MCT, Min-Min and Sufferage deterministic heuristics are presented in order to perform a comparative analysis. After that, the tables report the best and average makespan results achieved in 50 independent executions of pCHC, the standard deviation on the makespan values, and the improvement factor (impr., in percentage) over the best makespan result obtained with the deterministic heuristics.

Instance	MCT	Min-Min	Sufferage	pCHC			
				best	avg.	σ	impr.
A.u_c_hihi	32832740.0	22508064.0	30004648.0	20327924.0	20510300.9	0.14%	**9.69%**
A.u_c_hilo	3245777.0	2255966.3	2816620.5	2048582.7	2058352.2	0.16%	**9.19%**
A.u_c_lohi	3058.7	2155.0	2716.0	1956.7	2000.0	0.19%	**9.20%**
A.u_c_lolo	323.9	225.9	288.5	207.5	217.8	0.10%	**8.13%**
A.u_i_hihi	7567147.0	6367767.5	5601367.0	5169960.5	5244046.9	0.24%	**7.70%**
A.u_i_hilo	713132.4	641438.4	533545.2	490280.3	492699.4	0.11%	**8.11%**
A.u_i_lohi	754.1	664.7	551.7	518.2	523.6	0.15%	**8.32%**
A.u_i_lolo	73.4	63.7	55.4	50.6	51.7	0.19%	**8.63%**
A.u_s_hihi	19008366.0	14125880.0	16939632.0	12243560.0	12439843.1	0.10%	**13.33%**
A.u_s_hilo	1825499.9	1319050.5	1603158.5	1187506.4	1214303.0	0.20%	**9.97%**
A.u_s_lohi	1822.0	1380.5	1681.4	1186.8	1199.2	0.30%	**14.03%**
A.u_s_lolo	194.2	138.7	167.2	122.4	126.5	0.12%	**11.77%**
B.u_c_hihi	9478168.0	6708228.0	8514663.0	6169823.0	6200118.0	0.11%	**8.03%**
B.u_c_hilo	97584.4	66684.5	84876.8	61114.7	61390.1	0.10%	**8.35%**
B.u_c_lohi	333497.6	232011.8	296032.9	215149.2	218124.8	0.24%	**7.27%**
B.u_c_lolo	3402.3	2386.3	3105.7	2164.3	2208.4	0.11%	**9.30%**
B.u_i_hihi	2511410.8	2164576.5	1847652.5	1630288.6	1670112.7	0.11%	**11.76%**
B.u_i_hilo	22624.3	17083.1	16366.2	15121.5	15464.1	0.19%	**7.61%**
B.u_i_lohi	74041.1	56601.2	55083.2	49569.9	50128.2	0.13%	**10.01%**
B.u_i_lolo	743.8	585.0	537.1	496.1	507.4	0.10%	**7.64%**
B.u_s_hihi	5458156.0	3967265.5	4714483.5	3393010.2	3430218.1	0.10%	**14.47%**
B.u_s_hilo	55659.5	40691.6	50884.3	35988.4	36515.6	0.27%	**11.56%**
B.u_s_lohi	176744.7	135624.6	155599.9	115179.2	118070.3	0.19%	**13.08%**
B.u_s_lolo	1888.6	1333.2	1646.6	1191.7	1230.3	0.15%	**10.61%**

Table 6.14: pCHC results for new HCSP instances with dimension 1024×32.

Instance	MCT	Min-Min	Sufferage	pCHC			
				best	avg.	σ	impr.
A.u_c_hihi	28519530.0	19552222.0	25579850.0	18110479.1	18218285.6	0.65%	**7.37%**
A.u_c_hilo	2745652.5	1873134.3	2478699.3	1748509.2	1760141.2	0.47%	**6.65%**
A.u_c_lohi	2858.8	1924.7	2539.2	1798.4	1804.9	0.19%	**6.56%**
A.u_c_lolo	279.9	191.7	249.8	177.6	178.1	0.16%	**7.35%**
A.u_i_hihi	3900502.5	3248935.5	3218272.5	2506258.5	2546459.7	0.25%	**22.12%**
A.u_i_hilo	409815.0	365828.6	315267.5	272741.3	273876.3	0.32%	**13.49%**
A.u_i_lohi	385.2	320.9	312.5	266.3	267.5	0.28%	**14.80%**
A.u_i_lolo	41.8	32.3	29.5	26.4	26.5	0.29%	**10.48%**
A.u_s_hihi	16498318.0	11245679.0	13890956.0	9756499.7	9821934.5	0.58%	**13.24%**
A.u_s_hilo	1432291.0	1042948.5	1307394.3	924094.9	937998.8	1.44%	**11.40%**
A.u_s_lohi	1512.6	1056.0	1354.1	947.1	952.3	0.34%	**10.31%**
A.u_s_lolo	163.1	114.6	142.3	99.6	100.4	0.47%	**13.15%**
B.u_c_hihi	8236068.5	5564664.0	7560320.5	5290128.2	5300316.1	0.14%	**4.93%**
B.u_c_hilo	87265.9	59352.8	79079.2	55316.2	55343.1	0.06%	**6.80%**
B.u_c_lohi	281350.6	190842.4	253468.1	177063.4	177612.4	0.28%	**7.22%**
B.u_c_lolo	2882.3	1927.7	2613.8	1814.7	1818.3	0.19%	**5.86%**
B.u_i_hihi	1204421.0	929295.8	879421.3	770110.6	774993.0	0.47%	**12.43%**
B.u_i_hilo	11715.7	10318.4	9047.6	7906.5	7932.9	0.53%	**12.61%**
B.u_i_lohi	40528.6	34071.0	32073.6	26941.2	27207.3	0.61%	**16.00%**
B.u_i_lolo	413.9	355.7	299.4	262.4	264.7	0.26%	**12.36%**
B.u_s_hihi	4715914.0	3293157.0	4121618.8	2910507.6	2923857.1	0.34%	**11.62%**
B.u_s_hilo	47549.7	33445.4	41777.5	29442.2	29518.6	0.22%	**11.97%**
B.u_s_lohi	159401.9	111237.4	142534.7	98607.0	98758.3	0.19%	**11.35%**
B.u_s_lolo	1615.2	1163.8	1474.0	1014.3	1019.7	0.30%	**12.85%**

Table 6.15: pCHC results for new HCSP instances with dimension 2048×64.

Instance	MCT	Min-Min	Sufferage	pCHC			
				best	**avg.**	σ	impr.
A.u_c_hihi	24968242.0	16711134.0	23173816.0	15722681.0	15760840.0	0.16%	**9.69%**
A.u_c_hilo	2466416.3	1649763.5	2240514.0	1562810.9	1565580.1	0.17%	**9.19%**
A.u_c_lohi	2512.0	1635.3	2248.6	1540.9	1545.1	0.13%	**9.20%**
A.u_c_lolo	247.3	166.9	223.9	155.7	156.2	0.19%	**8.13%**
A.u_i_hihi	1939731.8	1666126.5	1575787.6	1309493.5	1331529.0	0.37%	**7.70%**
A.u_i_hilo	203714.6	177692.2	154506.9	137158.4	139250.8	0.17%	**8.11%**
A.u_i_lohi	203.5	188.0	165.6	136.1	137.7	0.21%	**8.32%**
A.u_i_lolo	20.8	19.4	15.2	13.7	13.7	0.27%	**8.63%**
A.u_s_hihi	13101840.0	8949853.0	11756833.0	8089853.5	8121957.0	0.29%	**13.33%**
A.u_s_hilo	1369277.1	930564.0	1215532.5	828912.4	834878.5	0.42%	**9.97%**
A.u_s_lohi	1310.7	927.9	1181.7	807.6	811.9	0.32%	**14.03%**
A.u_s_lolo	133.9	94.7	122.3	84.2	84.5	0.27%	**11.77%**
B.u_c_hihi	7715335.5	5059571.5	6912596.5	4767774.5	4789005.9	0.20%	**8.03%**
B.u_c_hilo	73858.5	49301.2	66003.5	46350.1	46470.8	0.14%	**8.35%**
B.u_c_lohi	253202.0	169495.3	230424.2	158780.8	159312.0	0.17%	**7.27%**
B.u_c_lolo	2464.7	1662.3	2263.6	1556.8	1562.2	0.16%	**9.30%**
B.u_i_hihi	630009.9	524174.1	472071.9	402182.1	405768.5	0.46%	**11.76%**
B.u_i_hilo	6333.5	5381.1	4964.7	4224.8	4252.2	0.23%	**7.61%**
B.u_i_lohi	21320.7	18772.1	15873.5	13847.8	13905.8	0.36%	**10.01%**
B.u_i_lolo	210.7	183.9	152.4	137.4	138.9	0.26%	**7.64%**
B.u_s_hihi	4065974.5	2843118.3	3551046.8	2508467.3	2524194.9	0.30%	**14.47%**
B.u_s_hilo	41297.3	27793.4	36605.5	25244.1	25346.6	0.43%	**11.56%**
B.u_s_lohi	131824.3	91523.0	116056.8	81118.5	81559.4	0.45%	**13.08%**
B.u_s_lolo	1358.2	921.8	1183.5	825.7	830.9	0.45%	**10.61%**

Table 6.16: pCHC results for new HCSP instances with dimension 4096×128.

Instance	MCT	Min-Min	Sufferage	pCHC			
				best	**avg.**	σ	impr.
A.u_c_hihi	22273440.0	14798376.0	20198762.0	14070023.0	14105642.9	0.15%	**7.37%**
A.u_c_hilo	2279612.5	1500181.5	2055377.3	1426068.0	1429947.9	0.09%	**6.65%**
A.u_c_lohi	2214.7	1456.5	2032.7	1384.8	1386.8	0.12%	**6.56%**
A.u_c_lolo	229.4	148.9	207.3	140.9	141.2	0.17%	**7.35%**
A.u_i_hihi	1075384.9	878829.5	788940.8	702540.6	715247.8	0.22%	**22.12%**
A.u_i_hilo	102423.2	85076.7	77317.0	70199.3	70648.3	0.61%	**13.49%**
A.u_i_lohi	102.2	96.1	82.6	71.0	73.5	0.32%	**14.80%**
A.u_i_lolo	11.6	8.8	8.0	7.1	7.3	0.17%	**10.48%**
A.u_s_hihi	11963559.0	8151522.0	10828664.0	7428847.5	7450818.7	0.32%	**13.24%**
A.u_s_hilo	1141591.6	787507.6	1047018.1	711087.9	714308.1	0.17%	**11.40%**
A.u_s_lohi	1165.5	796.9	1066.1	722.2	723.6	0.17%	**10.31%**
A.u_s_lolo	120.3	81.2	107.9	73.8	74.1	0.08%	**13.15%**
B.u_c_hihi	6880980.5	4460896.5	6251939.0	4254320.5	4261656.4	0.11%	**4.93%**
B.u_c_hilo	67167.0	43670.3	60967.2	41535.6	41614.2	0.10%	**6.80%**
B.u_c_lohi	225926.1	148102.7	203203.7	140752.1	140957.9	0.08%	**7.22%**
B.u_c_lolo	2214.8	1468.6	2000.6	1393.4	1396.6	0.12%	**5.86%**
B.u_i_hihi	313647.3	286800.2	248651.3	211439.3	216258.1	0.24%	**12.43%**
B.u_i_hilo	3029.4	2960.2	2496.7	2099.7	2147.5	0.22%	**12.61%**
B.u_i_lohi	10259.6	9496.4	7887.3	7017.2	7134.3	0.19%	**16.00%**
B.u_i_lolo	114.2	90.0	78.8	71.0	72.5	0.29%	**12.36%**
B.u_s_hihi	3461801.3	2411292.0	3137134.0	2155649.3	2167769.3	0.24%	**11.62%**
B.u_s_hilo	34836.5	23979.2	31826.8	21799.3	21873.5	0.22%	**11.97%**
B.u_s_lohi	117009.3	79291.5	106247.6	72303.5	72431.4	0.18%	**11.35%**
B.u_s_lolo	1167.2	807.2	1063.7	726.2	728.0	0.22%	**12.85%**

Table 6.17: pCHC results for new HCSP instances with dimension 8192×256.

The analysis of Tables 6.14 to 6.17 shows that pCHC achieved significant makespan improvements –ranging from 4.92% to 22.12%– over the best list scheduling heuristics results. pCHC takes advantage of the multiple search and the increased diversity provided by the subpopulation model to achieve high quality results. The standard deviation of the makespan values were very small (below 1.0%), demonstrating a high robustness behavior when solving the HCSP. Regarding inconsistent and semiconsistent instances, pCHC obtained a roughly 10% makespan improvement factor over the best traditional heuristic, even for the larger instances. The classic methods provide accurate schedules for inconsistent low-dimension HCSP instances, so the pCHC improvements were slightly below 10% for those scenarios. However, pCHC notably improves over the traditional methods for large inconsistent HCSP instances, achieving makespan reductions above **10%** for all problem instances, and a maximum of **22.12%** for `A.u_i_hihi` with dimension 2048×64. The opposite situation happens for consistent instances, where pCHC improvements diminished from nearly 10% to 5% as the instances dimension grows.

Table 6.18 and Figure 6.6 summarize the averaged pCHC improvements over the best traditional heuristic for each type of HCSP instance and dimension studied. Consistent instances are the most difficult to solve with pCHC, while accurate results were obtained for inconsistent and semiconsistent HCSP instances. The improvement values tend to slightly decrease as the instances grow, but they always remain above 5% for consistent, above 8% for inconsistent, and above 10% for semiconsistent instances.

model	**type**	**dimension**					**avg.**
		512×16	1024×32	2048×64	4096×128	8192×256	
Ali et al.	consistent	11.34%	9.05%	6.98%	5.93%	5.04%	**7.67%**
	inconsistent	9.85%	8.19%	15.22%	13.86%	11.46%	**11.72%**
	semiconsistent	14.31%	12.28%	12.03%	11.14%	9.28%	**11.81%**
Braun et al.	consistent	8.51%	8.24%	6.21%	6.11%	4.90%	**6.79%**
	inconsistent	7.53%	9.26%	13.35%	13.08%	12.93%	**11.23%**
	semiconsistent	11.32%	12.43%	11.95%	10.68%	9.64%	**11.20%**
	avg.	**10.48%**	**9.91%**	**10.96%**	**10.13%**	**8.88%**	

Table 6.18: pCHC improvements over traditional heuristics.

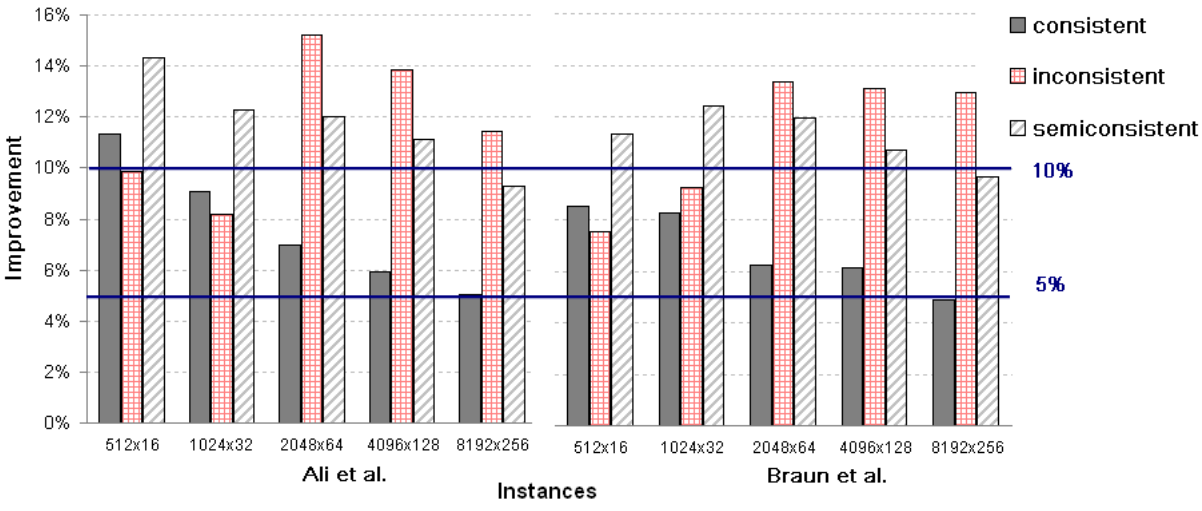


Figure 6.6: pCHC improvement over traditional heuristics.

The previous results show that the parallel version of the CHC algorithm is an accurate scheduler for the new large-dimension HCSP instances introduced in this work.

6.5.2 Parallel micro-CHC

Tables 6.19 to 6.22 present the results obtained when using pμ-CHC for solving the large HSCP instances with dimension 1024×32, 2048×64, 4096×128, and 8192×256, respectively. The tables report the best, average, and the standard deviation on the makespan values obtained in 50 independent executions of pμ-CHC for each problem instance studied. The results obtained with the MCT, Min-Min and Sufferage list scheduling heuristics, and the improvement factor (impr., in percentage) over the best deterministic heuristic results are also reported.

Instance	**MCT**	**Min-Min**	**Sufferage**	**pμ-CHC**			
				best	**avg.**	σ	impr.
A.u_c_hihi	32832740.0	22508064.0	26063096.0	19676858.0	19717711.4	0.11%	**12.58%**
A.u_c_hilo	3245777.0	2255966.3	2694595.0	1969980.0	1975398.9	0.11%	**12.68%**
A.u_c_lohi	3058.7	2155.0	2537.5	1887.3	1892.6	0.14%	**12.42%**
A.u_c_lolo	323.9	225.9	261.0	201.2	201.5	0.08%	**10.93%**
A.u_i_hihi	7567147.0	6367767.5	5601367.0	5126273.0	5147216.4	0.07%	**8.48%**
A.u_i_hilo	713132.4	641438.4	533545.2	485189.8	487879.4	0.19%	**9.06%**
A.u_i_lohi	754.1	664.7	551.7	513.8	516.8	0.24%	**6.87%**
A.u_i_lolo	73.4	63.7	55.4	50.2	50.4	0.20%	**9.39%**
A.u_s_hihi	19008366.0	14125880.0	14481880.0	11837170.0	11870719.2	0.15%	**16.20%**
A.u_s_hilo	1825499.9	1319050.5	1379341.3	1148940.7	1155387.1	0.25%	**12.90%**
A.u_s_lohi	1822.0	1380.5	1417.8	1152.5	1155.3	0.20%	**16.52%**
A.u_s_lolo	194.2	138.7	141.0	118.9	119.4	0.18%	**14.28%**
B.u_c_hihi	9478168.0	6708228.0	7874972.0	6049220.5	6052322.9	0.05%	**9.82%**
B.u_c_hilo	97584.4	66684.5	77250.5	59679.5	59730.6	0.06%	**10.50%**
B.u_c_lohi	333497.6	232011.8	272422.6	210005.1	210370.4	0.10%	**9.49%**
B.u_c_lolo	3402.3	2386.3	2826.9	2100.0	2103.3	0.10%	**12.00%**
B.u_i_hihi	2511410.8	2164576.5	1847652.5	1616697.4	1621628.0	0.11%	**12.50%**
B.u_i_hilo	22624.3	17083.1	16366.2	14993.2	15047.8	0.26%	**8.39%**
B.u_i_lohi	74041.1	56601.2	55083.2	49060.5	49351.9	0.31%	**10.93%**
B.u_i_lolo	743.8	585.0	537.1	487.5	491.8	0.38%	**9.23%**
B.u_s_hihi	5458156.0	3967265.5	3969449.5	3255266.8	3272088.3	0.22%	**17.95%**
B.u_s_hilo	55659.5	40691.6	41551.2	34675.2	34747.2	0.19%	**14.79%**
B.u_s_lohi	176744.7	135624.6	132510.3	110749.7	111068.5	0.20%	**16.42%**
B.u_s_lolo	1888.6	1333.2	1403.3	1153.1	1158.0	0.24%	**13.51%**

Table 6.19: pμ-CHC results for new HCSP instances with dimension 1024×32.

The analysis of Tables 6.19 to 6.22 shows that pμ-CHC is able to obtain significant better makespan results than the three deterministic list scheduling heuristics studied for all problem instances, while it specifically achieved large improvements for high-dimension HCSP instances. The makespan improvements (averaged per problem consistency and dimension) ranged from **7.5%** (consistent, 8192×256) to **16%** (semi-consistent, 2048×64). In addition, pμ-CHC maintained low values of standard deviation in the makespan results (below 0.5%), suggesting that the robust behavior exhibited in low-dimension HCSP instances also extends to large problem instances.

Instance	MCT	Min-Min	Sufferage	pμ-CHC			
				best	avg.	σ	impr.
A.u_c_hihi	28519530.0	19552222.0	25579850.0	17474552.0	17495744.9	0.11%	**10.63%**
A.u_c_hilo	2745652.5	1873134.3	2478699.3	1692750.0	1699639.9	0.11%	**9.63%**
A.u_c_lohi	2858.8	1924.7	2539.2	1731.9	1734.0	0.08%	**10.02%**
A.u_c_lolo	279.9	191.7	249.8	171.7	171.8	0.05%	**10.43%**
A.u_i_hihi	3900502.5	3248935.5	3218272.5	2477753.9	2492860.9	0.14%	**23.01%**
A.u_i_hilo	409815.0	365828.6	315267.5	272181.1	272529.4	0.17%	**13.67%**
A.u_i_lohi	385.2	320.9	312.5	265.6	266.2	0.20%	**15.01%**
A.u_i_lolo	41.8	32.3	29.5	26.3	26.4	0.19%	**10.85%**
A.u_s_hihi	16498318.0	11245679.0	13890956.0	9359727.3	9379560.0	0.13%	**16.77%**
A.u_s_hilo	1432291.0	1042948.5	1307394.3	878838.4	880125.4	0.13%	**15.74%**
A.u_s_lohi	1512.6	1056.0	1354.1	911.8	913.7	0.12%	**13.66%**
A.u_s_lolo	163.1	114.6	142.3	95.0	95.1	0.09%	**17.10%**
B.u_c_hihi	8236068.5	5564664.0	7560320.5	5085005.2	5092126.1	0.05%	**8.62%**
B.u_c_hilo	87265.9	59352.8	79079.2	53236.9	53306.3	0.06%	**10.30%**
B.u_c_lohi	281350.6	190842.4	253468.1	170659.4	170940.3	0.10%	**10.58%**
B.u_c_lolo	2882.3	1927.7	2613.8	1749.4	1754.7	0.10%	**9.25%**
B.u_i_hihi	1204421.0	929295.8	879421.3	763701.5	766428.7	0.15%	**13.16%**
B.u_i_hilo	11715.7	10318.4	9047.6	7859.0	7913.4	0.17%	**13.14%**
B.u_i_lohi	40528.6	34071.0	32073.6	26769.6	26973.5	0.27%	**16.54%**
B.u_i_lolo	413.9	355.7	299.4	261.8	263.2	0.26%	**12.56%**
B.u_s_hihi	4715914.0	3293157.0	4121618.8	2789531.9	2796655.7	0.18%	**15.29%**
B.u_s_hilo	47549.7	33445.4	41777.5	28170.9	28209.7	0.11%	**15.77%**
B.u_s_lohi	159401.9	111237.4	142534.7	93798.0	93997.5	0.17%	**15.68%**
B.u_s_lolo	1615.2	1163.8	1474.0	969.7	972.9	0.17%	**16.68%**

Table 6.20: pμ-CHC results for new HCSP instances with dimension 2048×64.

Instance	MCT	Min-Min	Sufferage	pμ-CHC			
				best	avg.	σ	impr.
A.u_c_hihi	24968242.0	16711134.0	23173816.0	15260752.4	15277595.2	0.05%	**8.68%**
A.u_c_hilo	2466416.3	1649763.5	2240514.0	1520225.1	1521480.9	0.04%	**7.85%**
A.u_c_lohi	2512.0	1635.3	2248.6	1493.8	1495.0	0.05%	**8.65%**
A.u_c_lolo	247.3	166.9	223.9	151.1	151.2	0.05%	**9.47%**
A.u_i_hihi	1939731.8	1666126.5	1575787.6	1295054.0	1312530.8	0.38%	**17.82%**
A.u_i_hilo	203714.6	177692.2	154506.9	135985.3	137480.8	0.14%	**11.99%**
A.u_i_lohi	203.5	188.0	165.6	135.3	136.6	0.19%	**18.30%**
A.u_i_lolo	20.8	19.4	15.2	13.6	13.8	0.28%	**10.53%**
A.u_s_hihi	13101840.0	8949853.0	11756833.0	7831962.8	7848970.9	0.13%	**12.49%**
A.u_s_hilo	1369277.1	930564.0	1215532.5	799499.4	801468.6	0.12%	**14.08%**
A.u_s_lohi	1310.7	927.9	1181.7	778.3	778.8	0.05%	**16.12%**
A.u_s_lolo	133.9	94.7	122.3	81.6	81.7	0.13%	**13.83%**
B.u_c_hihi	7715335.5	5059571.5	6912596.5	4649566.5	4651738.2	0.04%	**8.10%**
B.u_c_hilo	73858.5	49301.2	66003.5	45142.7	45190.5	0.06%	**8.43%**
B.u_c_lohi	253202.0	169495.3	230424.2	154504.7	154627.4	0.04%	**8.84%**
B.u_c_lolo	2464.7	1662.3	2263.6	1516.2	1517.4	0.05%	**8.79%**
B.u_i_hihi	630009.9	524174.1	472071.9	398655.1	403433.1	0.21%	**15.55%**
B.u_i_hilo	6333.5	5381.1	4964.7	4174.5	4238.4	0.24%	**15.92%**
B.u_i_lohi	21320.7	18772.1	15873.5	13614.6	13876.5	0.13%	**14.23%**
B.u_i_lolo	210.7	183.9	152.4	136.3	137.9	0.31%	**10.56%**
B.u_s_hihi	4065974.5	2843118.3	3551046.8	2437604.5	2440304.5	0.05%	**14.26%**
B.u_s_hilo	41297.3	27793.4	36605.5	24353.7	24397.9	0.10%	**12.38%**
B.u_s_lohi	131824.3	91523.0	116056.8	78296.8	78455.5	0.10%	**14.45%**
B.u_s_lolo	1358.2	921.8	1183.5	800.7	801.2	0.06%	**13.14%**

Table 6.21: pμ-CHC results for new HCSP instances with dimension 4096×128.

Instance	MCT	Min-Min	Sufferage	pμ-CHC			
				best	avg.	σ	impr.
A.u_c_hihi	22273440.0	14798376.0	20198762.0	13708686.6	13712470.2	0.02%	**7.36%**
A.u_c_hilo	2279612.5	1500181.5	2055377.3	1388689.3	1390076.8	0.04%	**7.43%**
A.u_c_lohi	2214.7	1456.5	2032.7	1344.2	1344.8	0.03%	**7.71%**
A.u_c_lolo	229.4	148.9	207.3	136.6	136.7	0.03%	**8.26%**
A.u_i_hihi	1075384.9	878829.5	788940.8	690223.8	693548.6	0.18%	**12.51%**
A.u_i_hilo	102423.2	85076.7	77317.0	68428.1	70310.8	0.14%	**11.50%**
A.u_i_lohi	102.2	96.1	82.6	68.9	71.4	0.46%	**16.59%**
A.u_i_lolo	11.6	8.8	8.0	6.9	7.1	0.22%	**13.75%**
A.u_s_hihi	11963559.0	8151522.0	10828664.0	7112313.0	7119414.4	0.06%	**12.75%**
A.u_s_hilo	1141591.6	787507.6	1047018.1	685350.9	686178.8	0.08%	**12.97%**
A.u_s_lohi	1165.5	796.9	1066.1	691.5	692.1	0.05%	**13.23%**
A.u_s_lolo	120.3	81.2	107.9	70.9	71.0	0.05%	**12.68%**
B.u_c_hihi	6880980.5	4460896.5	6251939.0	4136265.2	4137730.4	0.02%	**7.28%**
B.u_c_hilo	67167.0	43670.3	60967.2	40410.0	40425.6	0.03%	**7.47%**
B.u_c_lohi	225926.1	148102.7	203203.7	136499.6	136582.1	0.04%	**7.83%**
B.u_c_lolo	2214.8	1468.6	2000.6	1357.0	1357.4	0.02%	**7.60%**
B.u_i_hihi	313647.3	286800.2	248651.3	205347.6	212492.0	0.29%	**17.42%**
B.u_i_hilo	3029.4	2960.2	2496.7	2043.0	2108.9	0.34%	**18.17%**
B.u_i_lohi	10259.6	9496.4	7887.3	6812.3	7058.0	0.31%	**13.63%**
B.u_i_lolo	114.2	90.0	78.8	69.2	71.5	0.31%	**12.18%**
B.u_s_hihi	3461801.3	2411292.0	3137134.0	2087688.3	2088934.2	0.04%	**13.42%**
B.u_s_hilo	34836.5	23979.2	31826.8	21004.6	21045.2	0.08%	**12.40%**
B.u_s_lohi	117009.3	79291.5	106247.6	69347.8	69454.9	0.07%	**12.54%**
B.u_s_lolo	1167.2	807.2	1063.7	696.3	697.2	0.07%	**13.74%**

Table 6.22: pμ-CHC results for new HCSP instances with dimension 8192×256.

Figure 6.7 summarizes the pμ-CHC improvement factors with respect to the Min-Min and Sufferage deterministic scheduling methods, regarding the problem dimension and the consistency classification. The results show that the makespan improvement factors are around **15%** with respect to the Min-Min heuristic, and around **20%** (more than 25% for the largest instances) with respect to the Sufferage heuristic.

dimension	Min-Min	Sufferage
512×16	14.10%	17.47%
1024×32	15.42%	16.98%
2048×64	15.86%	26.67%
4096×128	15.95%	26.76%
8192×256	15.27%	27.35%

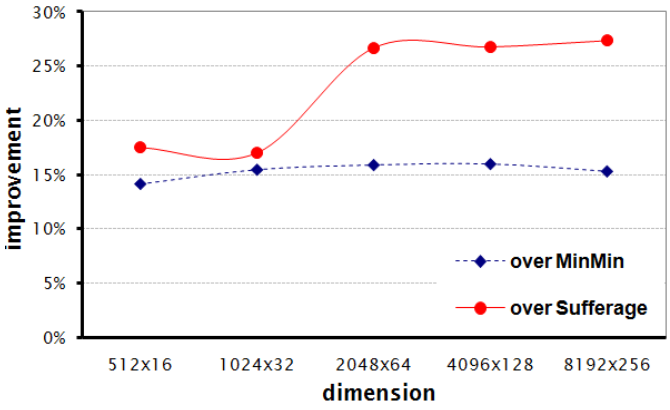


Figure 6.7: pμ-CHC improvements with respect to deterministic heuristics.

Table 6.23 presents the averaged improvements when using pμ-CHC with respect to the best deterministic heuristic for each problem dimension, consistency type, and heterogeneity model studied.

model	type	dimension					avg.
		512×16	1024×32	2048×64	4096×128	8192×256	
Ali	consistent	13.34%	12.15%	10.18%	8.66%	7.68%	**10.40%**
	inconsistent	10.97%	8.44%	15.63%	14.64%	13.70%	**12.68%**
	semiconsistent	16.62%	14.97%	15.81%	14.14%	12.92%	**14.89%**
Braun	consistent	9.19%	10.45%	9.69%	8.54%	7.54%	**9.08%**
	inconsistent	7.77%	10.27%	13.85%	14.06%	15.35%	**12.26%**
	semiconsistent	12.28%	16.15%	15.86%	13.56%	13.03%	**14.17%**
	avg.	**11.70%**	**12.07%**	**13.50%**	**12.27%**	**11.70%**	

Table 6.23: pμ-CHC improvements regarding the consistency classification.

Regarding the consistency classification, the pμ-CHC improvements over the best deterministic heuristic were always above **12%** for semiconsistent instances, and above **7%** for both consistent and inconsistent instances. The classic methods provide accurate schedules for low-dimension inconsistent HCSP instances, so the pμ-CHC improvements are roughly around 10% for those scenarios. However, pμ-CHC notably improves over the traditional methods for larger inconsistent HCSP instances, achieving makespan reductions above 13%, and a maximum of **23.01%** for `A.u_i_hihi` (dimension 2048×64). Figure 6.8 presents the graphical comparison of the average pμ-CHC improvements regarding each problem dimension, consistency type, and heterogeneity model studied. Three threshold lines (7%, 10%, and 12 %) are depicted to represent the lower levels of the improvement factors (averaged per problem dimension) for each consistency type.

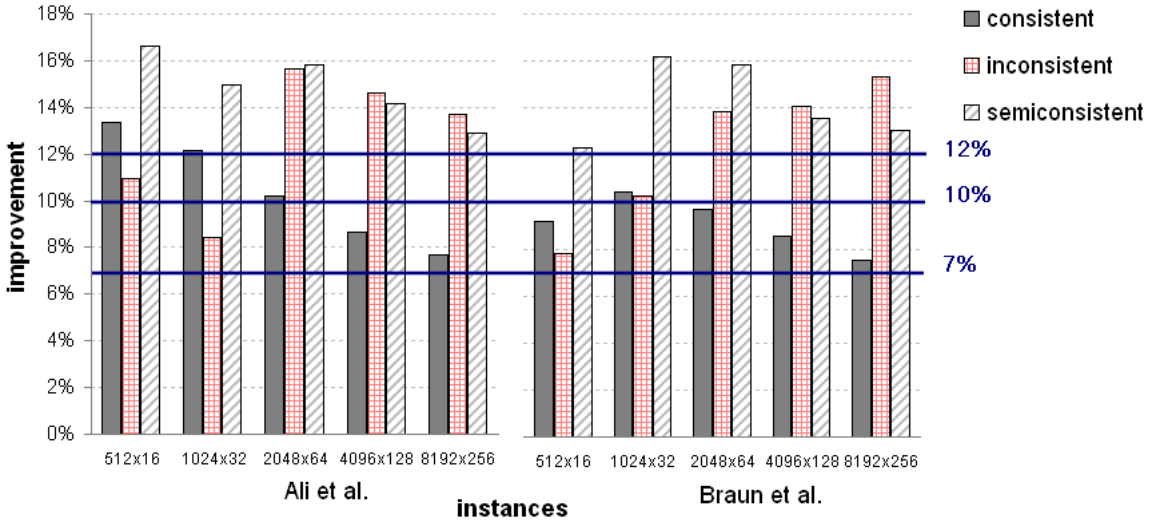


Figure 6.8: pμ-CHC improvements regarding the consistency classification.

pCHC and pμ-CHC comparison

Table 6.24 presents a comparative analysis of the results obtained with pCHC and pμ-CHC, oriented to study the effectiveness of the new pμ-CHC method to solve large HCSP instances. The table reports the best makespan values obtained with pCHC and pμ-CHC, and the improvement factors in the best and average makespan values obtained when using pμ-CHC over the pCHC results.

Instance	dimension: 1024×32				dimension: 2048×64			
	pCHC	pμ-CHC	impr.		pCHC	pμ-CHC	impr.	
	best	best	best	avg.	best	best	best	avg.
A.u_c_hihi	20327924.0	19676858.0	**3.20%**	**3.86%**	18110479.1	17474552.0	**3.51%**	**3.97%**
A.u_c_hilo	2048582.7	1969980.0	**3.84%**	**4.03%**	1748509.2	1692750.0	**3.19%**	**3.44%**
A.u_c_lohi	1956.7	1887.3	**3.55%**	**5.37%**	1798.4	1731.9	**3.70%**	**3.93%**
A.u_c_lolo	207.5	201.2	**3.04%**	**7.48%**	177.6	171.7	**3.32%**	**3.56%**
A.u_i_hihi	5169960.5	5126273.0	**0.85%**	**1.85%**	2506258.5	2477753.9	**1.14%**	**2.10%**
A.u_i_hilo	490280.3	485189.8	**1.04%**	**0.98%**	272741.3	272181.1	**0.21%**	**0.49%**
A.u_i_lohi	518.2	513.8	**0.85%**	**1.29%**	266.3	265.6	**0.26%**	**0.47%**
A.u_i_lolo	50.6	50.2	**0.79%**	**2.51%**	26.4	26.3	**0.38%**	**0.35%**
A.u_s_hihi	12243560.0	11837170.0	**3.32%**	**4.58%**	9756499.7	9359727.3	**4.07%**	**4.50%**
A.u_s_hilo	1187506.4	1148940.7	**3.25%**	**4.85%**	924094.9	878838.4	**4.90%**	**6.17%**
A.u_s_lohi	1186.8	1152.5	**2.89%**	**3.66%**	947.1	911.8	**3.73%**	**4.05%**
A.u_s_lolo	122.4	118.9	**2.86%**	**5.61%**	99.6	95.0	**4.62%**	**5.26%**
B.u_c_hihi	6169823.0	6049220.5	**1.95%**	**2.38%**	5290128.2	5085005.2	**3.88%**	**3.93%**
B.u_c_hilo	61114.7	59679.5	**2.35%**	**2.70%**	55316.2	53236.9	**3.76%**	**3.68%**
B.u_c_lohi	215149.2	210005.1	**2.39%**	**3.56%**	177063.4	170659.4	**3.62%**	**3.76%**
B.u_c_lolo	2164.3	2100.0	**2.97%**	**4.76%**	1814.7	1749.4	**3.60%**	**3.50%**
B.u_i_hihi	1630288.6	1616697.4	**0.83%**	**2.90%**	770110.6	763701.5	**0.83%**	**1.11%**
B.u_i_hilo	15121.5	14993.2	**0.85%**	**2.69%**	7906.5	7859.0	**0.60%**	**0.25%**
B.u_i_lohi	49569.9	49060.5	**1.03%**	**1.55%**	26941.2	26769.6	**0.64%**	**0.86%**
B.u_i_lolo	496.1	487.5	**1.73%**	**3.07%**	262.4	261.8	**0.23%**	**0.55%**
B.u_s_hihi	3393010.2	3255266.8	**4.06%**	**4.61%**	2910507.6	2789531.9	**4.16%**	**4.35%**
B.u_s_hilo	35988.4	34675.2	**3.65%**	**4.84%**	29442.2	28170.9	**4.32%**	**4.43%**
B.u_s_lohi	115179.2	110749.7	**3.85%**	**5.93%**	98607.0	93798.0	**4.88%**	**4.82%**
B.u_s_lolo	1191.7	1153.1	**3.24%**	**5.88%**	1014.3	969.7	**4.40%**	**4.59%**

Instance	dimension: 4096×128				dimension: 8192×256			
	pCHC	pμ-CHC	impr.		pCHC	pμ-CHC	impr.	
	best	best	best	avg.	best	best	best	avg.
A.u_c_hihi	15722681.0	15260752.4	**2.94%**	**3.07%**	14070023.0	13708686.6	2.57%	2.79%
A.u_c_hilo	1562810.9	1520225.1	**2.72%**	**2.82%**	1426068.0	1388689.3	**2.62%**	**2.79%**
A.u_c_lohi	1540.9	1493.8	**3.06%**	**3.24%**	1384.8	1344.2	**2.93%**	**3.03%**
A.u_c_lolo	155.7	151.1	**2.95%**	**3.18%**	140.9	136.6	**3.05%**	**3.17%**
A.u_i_hihi	1309493.5	1295054.0	**1.10%**	**1.43%**	702540.6	690223.8	**1.75%**	**3.03%**
A.u_i_hilo	137158.4	135985.3	**0.86%**	**1.27%**	70199.3	68428.1	**2.52%**	**0.48%**
A.u_i_lohi	136.1	135.3	**0.59%**	**0.78%**	71.0	68.9	**2.96%**	**2.80%**
A.u_i_lolo	13.7	13.6	**0.73%**	**5.90%**	7.1	6.9	**2.82%**	**2.99%**
A.u_s_hihi	8089853.5	7831962.8	**3.19%**	**3.36%**	7428847.5	7112313.0	**4.26%**	**4.45%**
A.u_s_hilo	828912.4	799499.4	**3.55%**	**4.00%**	711087.9	685350.9	**3.62%**	**3.94%**
A.u_s_lohi	807.6	778.3	**3.63%**	**4.07%**	722.2	691.5	**4.25%**	**4.36%**
A.u_s_lolo	84.2	81.6	**3.09%**	**3.32%**	73.8	70.9	**3.93%**	**4.13%**
B.u_c_hihi	4767774.5	4649566.5	**2.48%**	**2.87%**	4254320.5	4136265.2	**2.77%**	**2.91%**
B.u_c_hilo	46350.1	45142.7	**2.60%**	**2.75%**	41535.6	40410.0	**2.71%**	**2.86%**
B.u_c_lohi	158780.8	154504.7	**2.69%**	**2.94%**	140752.1	136499.6	**3.02%**	**3.10%**
B.u_c_lolo	1556.8	1516.2	**2.61%**	**2.87%**	1393.4	1357.0	**2.61%**	**2.81%**
B.u_i_hihi	402182.1	398655.1	**0.88%**	**0.58%**	211439.3	205347.6	**2.88%**	**1.74%**
B.u_i_hilo	4224.8	4174.5	**1.19%**	**0.32%**	2099.7	2043.0	**2.70%**	**1.80%**
B.u_i_lohi	13847.8	13614.6	**1.68%**	**0.21%**	7017.2	6812.3	**2.92%**	**1.07%**
B.u_i_lolo	137.4	136.3	**0.80%**	**0.74%**	71.0	69.2	**2.54%**	**1.32%**
B.u_s_hihi	2508467.3	2437604.5	**2.82%**	**3.32%**	2155649.3	2087688.3	**3.15%**	**3.64%**
B.u_s_hilo	25244.1	24353.7	**3.53%**	**3.74%**	21799.3	21004.6	**3.65%**	**3.79%**
B.u_s_lohi	81118.5	78296.8	**3.48%**	**3.81%**	72303.5	69347.8	**4.09%**	**4.11%**
B.u_s_lolo	825.7	800.7	**3.03%**	**3.58%**	726.2	696.3	**4.12%**	**4.23%**

Table 6.24: pCHC and pμ-CHC comparative results for new HCSP instances.

The analysis of Table 6.24 shows that pμ-CHC consistently improves over the previous best-known results for the new test suite of HCSP instances, obtained with pCHC. The makespan improvements ranged from 0.21% to **4.90%**, and they tend to increase as the problem instances grow: the average improvement factor increases from 1.61% for the smaller instances to **3.09%** for HCSP instances with dimension 8192×256.

The previous results demonstrate the usefulness of the improved local search provided by the randomized PALS operator (the main algorithmic difference between pμ-CHC and pCHC). The specific local search procedure designed to introduce diversity in the population allows improving over the results of pCHC in more than a 3% improvement factor in the makespan values.

Table 6.25 summarizes the improvements obtained by pμ-CHC with respect to pCHC, averaged per problem dimension, and Figure 6.9 presents a graphical comparison of the overall makespan improvements.

dimension	best	average
512×16	1.61%	2.10%
1024×32	2.43%	3.79%
2048×64	2.83%	3.09%
4096×128	2.34%	2.68%
8192×256	3.11%	2.97%

Table 6.25: pμ-CHC improvements with respect to pCHC.

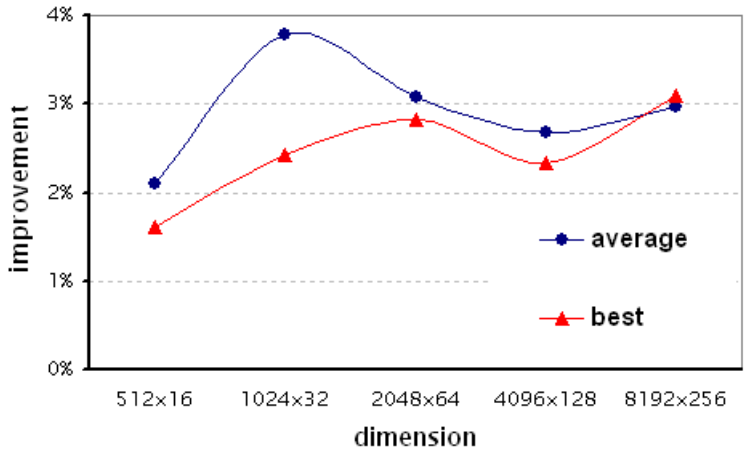


Figure 6.9: pμ-CHC improvements with respect to pCHC.

Regarding the consistency classification, pμ-CHC obtained slight improvements with respect to pCHC for inconsistent instances (around 1%), but the improvements were more significant in consistent (over 3%) and semi-consistent (over 4%) scenarios. The previously commented results demonstrate that pμ-CHC overcomes the problem of pCHC, which was unable to efficiently deal with large structured scenarios modeled by consistent HCSP instances.

As well as for the set of low-dimension problem instances by Braun et al. (2001), the previously presented results show that the parallel micro-CHC algorithm is the currently state-of-the-art scheduler for the large dimension HCSP instances studied.

6.6 Comparison with lower bounds for the preemptive HCSP

Due to the high computational complexity of the problem, the non-preemptive HSCP cannot be solved in reasonable execution times using exact methods. However, a lower bound for the makespan value can be computed by solving the linear relaxation for the preemptive case of the problem. Under the preemption hypothesis, the scheduler can temporarily interrupt a task and continue its execution at a later time, without additional costs for the context switch procedure. In this (unrealistic) situation, an optimal HCSP solution has all machines with the same value of local makespan, which corresponds to the optimal makespan of the schedule.

The HCSP linear relaxation was solved using `GLPSOL`, the free linear programming solver included in `GLPK`, the GNU Linear Programming Kit (Makhorin, 2006), for the small problem instances studied (dimension 512×12 and 1024×32). The high execution times required by `GLPSOL` make impractical to compute the lower bounds for larger problem instances, so a commercial version of `CPLEX` (ILOG, 2006) was used for large HCSP instances (dimension 2048×64, 4096×128, and 8192×256).

`GLPSOL` and `CPLEX` use the revised simplex method and the primal-dual interior point method (Darst, 1991) for solving non-integer optimization problems such as the preemptive version of the HCSP. `GLPSOL` requires a problem model specification prepared in the language `GMPL` (GNU MathProg Language), which is a subset of (and so, shares the syntax with) the well-known high-level programming language for mathematical optimization `AMPL` (Fourer et al., 2002). `GLPSOL` also requires a problem scenario specification written in `AMPL`, following a predefined format.

Algorithm 9 presents the linear programming (LP) model for the preemptive version of HCSP written in `GMPL/AMPL`.

Algorithm 9 LP model for the preemptive version of HCSP in GMPL/AMPL.

```
   // Variable definitions
1: set ST;
2: set SM;
3: param ETC{t in ST, m in SM};
4: var z;
5: var x{t in ST, m in SM} >= 0;
   // Objective function
6: minimize makespan :  z;
   // Constraints
7: s.t.  assign{t in ST}:  sum{m in SM} x[t,m] == 1;
8: s.t.  defmakespan{m in SM}:  z >= sum{t in ST} x[t,m] * ETC[t,m];
9: end;
```

The `GMPL/AMPL` model for the preemptive version of HCSP considers two sets: the set of tasks (`ST`) and the set of machines (`SM`). The parameters of the model are the ETC values (a matrix with $N \times M$ elements), the variable `z` (vector) –which represents the local makespan for each machine–, and the decision variables `x` (matrix) used to indicate the task-to-machine assignments. The objective function to minimize is the makespan of the schedule, defined as the maximum value of `z` for all machines in `SM`. The integrity constraint in line 7 assures that each task in `ST` is executed once.

Figure 6.10 presents an example of scenario specification for the HCSP, where the `data` section defines the problem data (tasks and machines) and the `param` section defines the scenario parameters (in this case, the ETC for each task on each machine, written in consecutive lines). The problem model and the scenario specification written in `AMPL` are also supported by `CPLEX`.

```
data;
set T:= 0 1 2 [...]  512;
set M:= 0 1 2 [...]  16;

param ETC:=
0 0 132189.530000
0 2 179199.270000
  ...
  ...
;
```

Figure 6.10: HCSP scenario specification in GMPL/AMPL.

The computed lower bounds are useful to determine the accuracy of the results achieved using the EAs proposed in this work for solving the HCSP. By computing the relative gap values between the EA's result and the lower bound (LB) for each problem instance (GAP(LB), defined by Equation 6.1), the distance between the EA result and the optimum makespan value (GAP(optimum)) can be estimated. The distance is bounded by the computed GAP(LB), since the optimum makespan value lays between the computed lower bound and the result obtained by the EA (see a graphical representation in Figure 6.11).

$$GAP(LB) = \frac{result - LB}{LB} \tag{6.1}$$

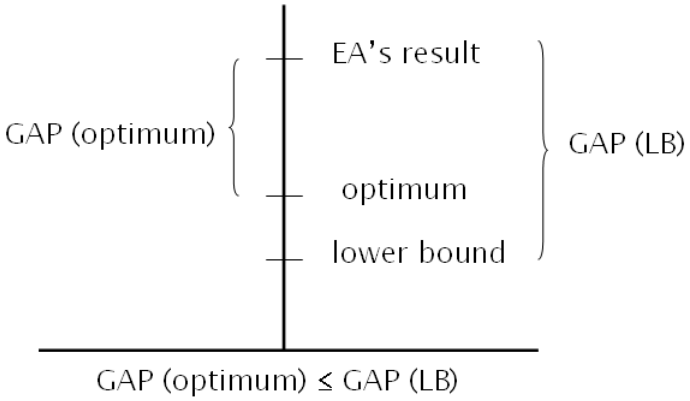


Figure 6.11: Relationship between GAP(optimum) and GAP(LB).

6.6.1 Small HCSP instances

Table 6.26 shows a comparison between the best and average makespan results achieved by pμ-CHC (the best algorithm among the studied methods) and the lower bounds (LB) computed for the preemptive case computed for the small-sized (dimension 512×16) HCSP instances faced in this work (generated using the ETC parametrization from Ali et al., and proposed by Braun et al.). The table also reports the relative gap values (GAP(LB)) for the best and average results achieved for each problem instance with respect to the correspondent lower bound.

random instances generated with the ETC parametrization from Ali et al. (2000)							
Instance	**LB**	**pμ-CHC (best)**	**GAP (LB)**	**avg. GAP(LB)**	**pμ-CHC (avg.)**	**GAP (LB)**	**avg. GAP(LB)**
A.u_c_hihi	21625636.6	21723328.0	**0.45%**	**0.56%**	21790559.1	**0.76%**	**0.78%**
A.u_c_hilo	2375833.1	2390466.5	**0.62%**		2394957.0	**0.80%**	
A.u_c_lohi	2247.8	2261.7	**0.62%**		2265.6	**0.79%**	
A.u_c_lolo	229.4	230.7	**0.55%**		231.2	**0.78%**	
A.u_i_hihi	8997802.5	9091042.0	**1.04%**	**1.09%**	9118626.4	**1.34%**	**1.52%**
A.u_i_hilo	877013.6	886644.2	**1.10%**		890981.1	**1.59%**	
A.u_i_lohi	954.7	964.4	**1.01%**		969.2	**1.51%**	
A.u_i_lolo	99.1	100.4	**1.23%**		100.8	**1.64%**	
A.u_s_hihi	16115868.3	16260961.5	**0.90%**	**0.96%**	16321566.5	**1.28%**	**1.33%**
A.u_s_hilo	1509214.7	1523632.3	**0.96%**		1528680.6	**1.29%**	
A.u_s_lohi	1455.3	1470.0	**1.01%**		1475.7	**1.40%**	
A.u_s_lolo	155.1	156.6	**0.98%**		157.2	**1.34%**	
de-facto standard benchmark instances from Braun et al. (2001)							
Instance	**LB**	**pμ-CHC (best)**	**GAP (LB)**	**avg. GAP(LB)**	**pμ-CHC (avg.)**	**GAP (LB)**	**avg. GAP(LB)**
u_c_hihi.0	7346524.2	7381570.0	**0.48%**	**0.38%**	7394702.7	**0.66%**	**0.50%**
u_c_hilo.0	152700.4	153105.4	**0.27%**		153193.7	**0.32%**	
u_c_lohi.0	238138.1	239260.0	**0.47%**		239706.2	**0.66%**	
u_c_lolo.0	5132.8	5147.9	**0.29%**		5152.3	**0.38%**	
u_i_hihi.0	2909326.6	2938380.8	**1.00%**	**0.73%**	2947896.4	**1.33%**	**1.00%**
u_i_hilo.0	73057.9	73387.0	**0.45%**		73531.4	**0.65%**	
u_i_lohi.0	101063.4	102050.6	**0.98%**		102402.8	**1.33%**	
u_i_lolo.0	2529.0	2541.4	**0.49%**		2547.1	**0.72%**	
u_s_hihi.0	4063563.7	4103500.3	**0.98%**	**0.82%**	4123537.3	**1.48%**	**1.17%**
u_s_hilo.0	95419.0	95787.4	**0.39%**		96020.5	**0.63%**	
u_s_lohi.0	120452.3	122083.3	**1.35%**		122744.4	**1.90%**	
u_s_lolo.0	3414.8	3433.5	**0.55%**		3438.3	**0.69%**	

Table 6.26: pμ-CHC comparison with lower bounds (dimension 512×16).

The makespan values reported in Table 6.26 show that pμ-CHC was able to achieve accurate results when compared with the (in the general case, unattainable) lower bounds for the preemptive case. The gaps were below **2%** for all instances, and below **1%** for half of the instances studied, assuring that there is a small distance between the obtained results and the optimal makespan value for each problem instance.

The comparison against the results obtained with the best deterministic heuristic and the computed lower bounds for the preemptive case demonstrate that pμ-CHC was able to achieve more than **90%** of the ideal improvement for the studied HCSP scenarios. Figure 6.12 summarizes the results for the problem instances by Braun et al. (2001), showing the improvements obtained by pμ-CHC with respect to both the Min-Min results and the ideal improvements considering the computed lower bounds for the preemptive case.

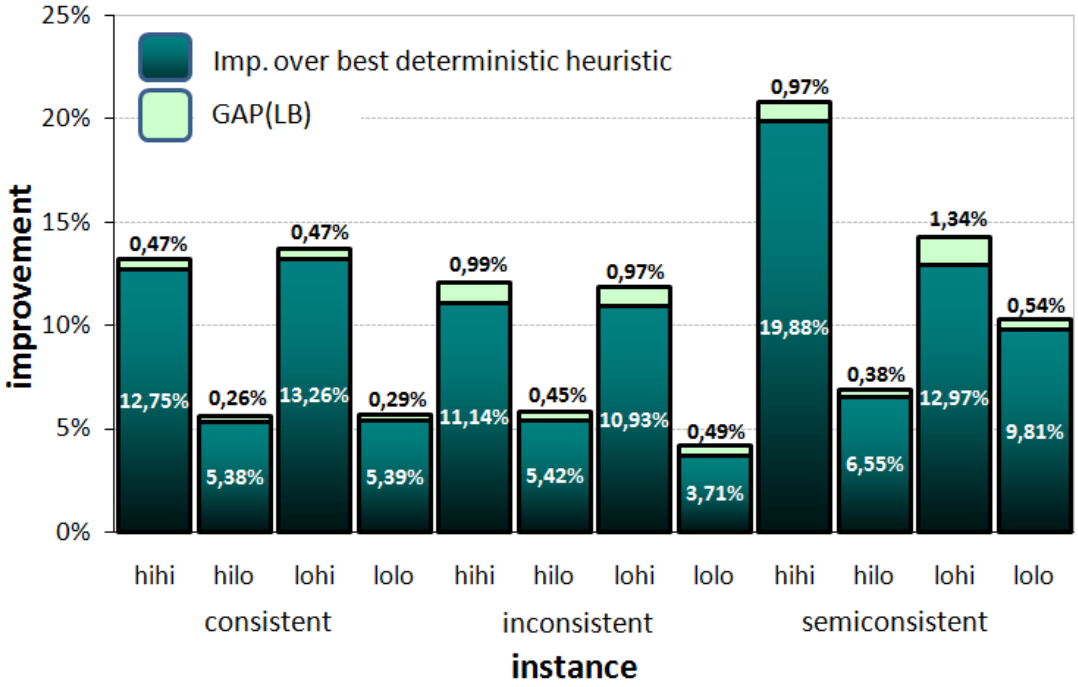


Figure 6.12: pμ-CHC improvements with respect to the best deterministic heuristic results and the computed lower bounds for HCSP instances by Braun et al. (2001).

6.6.2 New large HCSP instances

Table 6.27 presents the comparison between the average and best makespan results obtained using pμ-CHC and the lower bounds (LB) computed for the preemptive case computed for large-dimension HCSP instances. The table also reports the relative gap values for the best and average results achieved for each problem instance with respect to the correspondent lower bound (GAP(LB)).

The makespan values reported in Table 6.27 show that pμ-CHC was able to achieve accurate results when compared with the (in the general case, unattainable) lower bounds for the preemptive case. The gaps were below **9%** for all instances and dimensions, and below **5%** for 80 out of 96 instances studied, assuring that there is a small distance between the obtained results and the optimal makespan value for each problem instance.

6.6.3 Summary

Table 6.28 summarizes the comparison between the results achieved by pμ-CHC and the lower bounds computed for the preemptive case for each problem dimension studied. It reports the relative gap value of the best and average results achieved for each problem dimension with respect to the correspondent lower bound. The table also reports the improvements over the best deterministic heuristic and the percentage of the ideal improvement achieved by pμ-CHC, considering the computed lower bounds.

The values reported in Table 6.28 show that pμ-CHC is able to obtain accurate results when compared with the lower bounds for the preemptive case. For the 512×16 HCSP instances the gaps were below 1.5%, and below 1% for 11 out of 12 instances.

Instance	dimension: 1024×32					dimension: 2048×64				
	LB	pμ-CHC		GAP(LB)		LB	pμ-CHC		GAP(LB)	
		best	avg.	best	avg.		best	avg.	best	avg.
A.u_c_hihi	19449230.0	19676858.0	19717711.4	1.17%	1.38%	17141977.4	17474552.0	17495744.9	1.94%	2.06%
A.u_c_hilo	1951345.1	1969980.0	1975398.9	0.95%	1.23%	1664592.8	1692750.0	1699639.9	1.69%	2.11%
A.u_c_lohi	1866.4	1887.3	1892.6	1.12%	1.40%	1695.3	1731.9	1734.0	2.16%	2.28%
A.u_c_lolo	198.9	201.2	201.5	1.16%	1.31%	168.3	171.7	171.8	2.00%	2.06%
A.u_i_hihi	5012207.0	5126273.0	5147216.4	2.28%	2.69%	2366682.1	2477753.9	2492860.9	4.69%	5.33%
A.u_i_hilo	474404.6	485189.8	487879.4	2.27%	2.84%	260904.5	272181.1	272529.4	4.32%	4.46%
A.u_i_lohi	503.4	513.8	516.8	2.08%	2.67%	255.2	265.6	266.2	4.09%	4.33%
A.u_i_lolo	49.0	50.2	50.4	2.47%	2.88%	25.1	26.3	26.4	4.58%	4.98%
A.u_s_hihi	11553632.0	11837170.0	11870719.2	2.45%	2.74%	9050260.8	9359727.3	9379560.0	3.42%	3.64%
A.u_s_hilo	1126556.2	1148940.7	1155387.1	1.99%	2.56%	851399.9	878838.4	880125.4	3.22%	3.37%
A.u_s_lohi	1122.2	1152.5	1155.3	2.70%	2.95%	888.9	911.8	913.7	2.57%	2.79%
A.u_s_lolo	116.7	118.9	119.4	1.90%	2.33%	92.3	95.0	95.1	2.95%	3.06%
B.u_c_hihi	5980871.9	6049220.5	6052322.9	1.14%	1.19%	4975778.8	5085005.2	5092126.1	2.20%	2.34%
B.u_c_hilo	58942.5	59679.5	59730.6	1.25%	1.34%	52240.6	53236.9	53306.3	1.91%	2.04%
B.u_c_lohi	207892.8	210005.1	210370.4	1.02%	1.19%	167381.1	170659.4	170940.3	1.96%	2.13%
B.u_c_lolo	2078.0	2100.0	2103.3	1.06%	1.22%	1715.0	1749.4	1754.7	2.00%	2.31%
B.u_i_hihi	1567178.7	1616697.4	1621628.0	3.16%	3.47%	735101.5	763701.5	766428.7	3.89%	4.26%
B.u_i_hilo	14582.3	14993.2	15047.8	2.82%	3.19%	7536.3	7859.0	7913.4	4.28%	5.00%
B.u_i_lohi	47606.9	49060.5	49351.9	3.05%	3.67%	25681.2	26769.6	26973.5	4.24%	5.03%
B.u_i_lolo	477.4	487.5	491.8	2.11%	3.01%	250.5	261.8	263.2	4.53%	5.09%
B.u_s_hihi	3178482.2	3255266.8	3272088.3	2.42%	2.94%	2710023.7	2789531.9	2796655.7	2.93%	3.20%
B.u_s_hilo	33948.7	34675.2	34747.2	2.14%	2.35%	27268.0	28170.9	28209.7	3.31%	3.45%
B.u_s_lohi	108330.1	110749.7	111068.5	2.23%	2.53%	90727.3	93798.0	93997.5	3.38%	3.60%
B.u_s_lolo	1128.1	1153.1	1158.0	2.21%	2.65%	939.0	969.7	972.9	3.27%	3.61%

Instance	dimension: 4096×128					dimension: 8192×256				
	LB	pμ-CHC		GAP(LB)		LB	pμ-CHC		GAP(LB)	
		best	avg.	best	avg.		best	avg.	best	avg.
A.u_c_hihi	14829360.6	15260752.4	15277595.2	2.91%	3.02%	13338612.5	13708686.6	13712470.2	2.77%	2.80%
A.u_c_hilo	1478358.1	1520225.1	1521480.9	2.83%	2.92%	1352062.2	1388689.3	1390076.8	2.71%	2.81%
A.u_c_lohi	1452.5	1493.8	1495.0	2.84%	2.92%	1307.6	1344.2	1344.8	2.80%	2.84%
A.u_c_lolo	147.4	151.1	151.2	2.52%	2.58%	132.6	136.6	136.7	3.02%	3.10%
A.u_i_hihi	1231099.0	1295054.0	1312530.8	5.19%	6.61%	634712.8	690223.8	693548.6	8.75%	9.27%
A.u_i_hilo	128539.5	135985.3	137480.8	5.79%	6.96%	63530.6	68428.1	70310.8	7.71%	10.67%
A.u_i_lohi	127.6	135.3	136.6	6.02%	7.04%	63.7	68.9	71.4	8.17%	12.10%
A.u_i_lolo	12.9	13.6	13.8	5.51%	7.06%	6.4	6.9	7.1	7.86%	10.99%
A.u_s_hihi	7553763.4	7831962.8	7848970.9	3.68%	3.91%	6812019.5	7112313.0	7119414.4	4.41%	4.51%
A.u_s_hilo	768703.1	799499.4	801468.6	4.01%	4.26%	657203.0	685350.9	686178.8	4.28%	4.41%
A.u_s_lohi	748.5	778.3	778.8	3.99%	4.05%	661.9	691.5	692.1	4.47%	4.56%
A.u_s_lolo	78.4	81.6	81.7	4.14%	4.26%	67.9	70.9	71.0	4.36%	4.51%
B.u_c_hihi	4514305.9	4649566.5	4651738.2	3.00%	3.04%	4013215.4	4136265.2	4137730.4	3.07%	3.10%
B.u_c_hilo	44027.0	45142.7	45190.5	2.53%	2.64%	39256.7	40410.0	40425.6	2.94%	2.98%
B.u_c_lohi	150530.0	154504.7	154627.4	2.64%	2.72%	132570.1	136499.6	136582.1	2.96%	3.03%
B.u_c_lolo	1474.0	1516.2	1517.4	2.86%	2.94%	1319.5	1357.0	1357.4	2.84%	2.87%
B.u_i_hihi	374988.9	398655.1	403433.1	6.31%	7.59%	190045.7	205347.6	212492.0	8.05%	11.81%
B.u_i_hilo	3942.5	4174.5	4238.4	5.89%	7.51%	1894.2	2043.0	2108.9	7.85%	11.33%
B.u_i_lohi	12825.3	13614.6	13876.5	6.15%	8.20%	6311.4	6812.3	7058.0	7.94%	11.83%
B.u_i_lolo	128.8	136.3	137.9	5.79%	7.03%	64.1	69.2	71.5	7.92%	11.50%
B.u_s_hihi	2353555.3	2437604.5	2440304.5	3.57%	3.69%	2003494.4	2087688.3	2088934.2	4.20%	4.26%
B.u_s_hilo	23417.3	24353.7	24397.9	4.00%	4.19%	20179.1	21004.6	21045.2	4.09%	4.29%
B.u_s_lohi	75488.9	78296.8	78455.5	3.72%	3.93%	66458.3	69347.8	69454.9	4.35%	4.51%
B.u_s_lolo	771.8	800.7	801.2	3.75%	3.81%	668.4	696.3	697.2	4.17%	4.31%

Table 6.27: Comparison between pμ-CHC results and lower bounds calculated for the preemptive HCSP (large instances).

The gaps increase as the problem dimension grows, but even for the largest problem dimension tackled, the values are below **6.20%** (average) and **5.07%** (best). These results suggest that there is a small difference between the obtained results and the optimal makespan value for each problem instance, since the optimal makespan value lays between the computed LB and the pμ-CHC result.

dimension	GAP(LB)		imp.	% ideal imp.
	avg.	best		
512×16	1.05%	0.75%	14.10%	**94.93%**
1024×32	2.32%	1.96%	15.42%	**88.70%**
2048×64	3.44%	3.15%	15.86%	**82.18%**
4096×128	4.70%	4.15%	15.95%	**77.23%**
8192×256	6.18%	5.07%	15.27%	**71.18%**

Table 6.28: Summary: gaps between pμ-CHC results and lower bounds for the preemptive HCSP version.

Figure 6.13 presents a graphical summary of the improvements obtained by pμ-CHC over the best deterministic heuristic result and the comparison with the ideal improvement given by the computed lower bounds for each problem dimension.

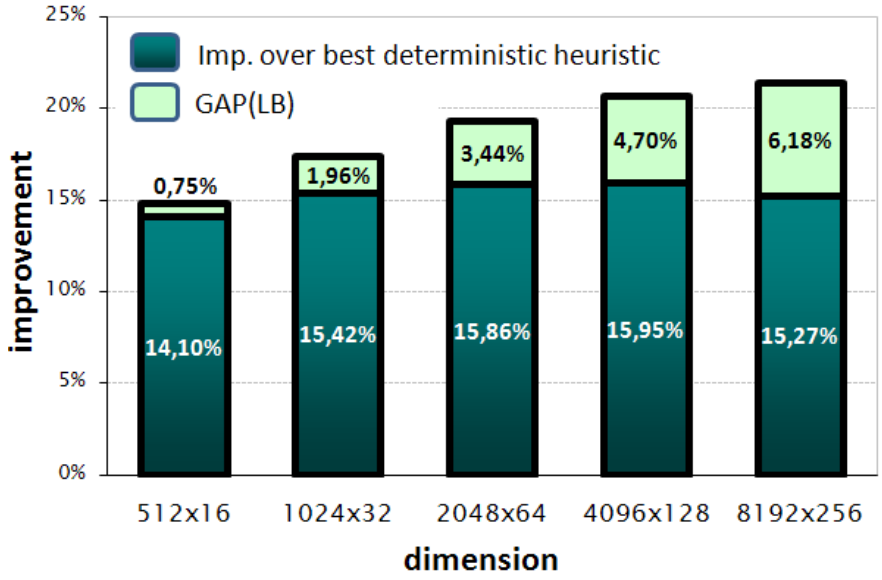


Figure 6.13: pμ-CHC improvements with respect to the best deterministic heuristic result and the computed lower bounds for the preemptive HCSP version.

6.7 Tracking the makespan evolution and the execution time

This subsection analyzes one important aspect of EAs: the objective function/fitness value evolution over the generations, and the trade-off between the solution quality obtained using the parallel EAs and the required execution time. When dealing with scheduling problems, the analysis of the variation of the makespan results with respect to the wall-clock time is crucial to determine the effectiveness of the scheduler to provide accurate plannings in reduced execution times.

6.7.1 Comparative makespan evolution of CHC algorithms

Figures 6.14 and 6.15 present the evolution of the best makespan values observed for the sequential and parallel CHC algorithms during representative executions over the HCSP instances `u_i_hihi.0` and `u_s_lohi.0` (the makespan value of the Min-Min and Sufferage solutions are included as a reference baseline). The slope of the curves show that the parallel models allows computing accurate schedules faster than the sequential CHC, which follows a more lethargic makespan evolution pattern. The same lethargic behavior was verified for the serial and parallel GA, thus they are omitted in the graphical analysis. These results confirms the ability of parallel EAs to obtain more accurate solutions than the sequential methods and also to reduce the required computing times.

In addition, the graphics show that pμ-CHC is able to find well-suited individuals in a short time, improving over traditional heuristics in a few seconds. The parallel CHC also has a good makespan evolution behavior, but it works a little bit slower when compared with pμ-CHC.

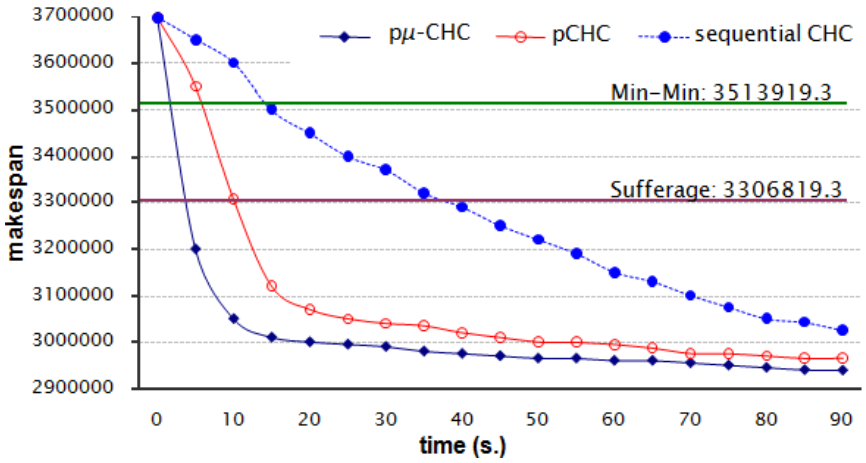


Figure 6.14: Makespan evolution for CHC algorithms on `u_i_hihi.0`.

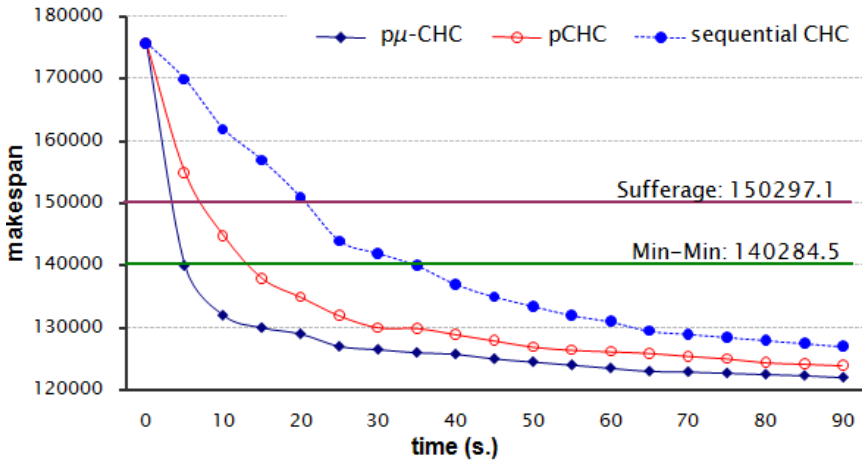


Figure 6.15: Makespan evolution for CHC algorithms on `u_s_lohi.0`.

Table 6.29 reports the variation of the average makespan values obtained in 20 executions of pCHC and pμ-CHC when using 10, 30, and 60 s. of execution time for three representative small scenarios with high task heterogeneity. The average improvements over the Min-Min results are also presented.

Instance	Min-Min	pCHC			pμ-CHC		
		10 s.	**30 s.**	**60 s.**	**10 s.**	**30 s.**	**60 s.**
u_c_lohi.0	275837.3	285717.4	254394.6	245213.7	248712.8	243103.7	241182.4
u_i_hihi.0	3513919.3	3306819.0	3042056.3	2990121.8	3051183.7	2980721.6	2952919.0
u_s_lohi.0	140284.5	144902.7	130109.2	126219.3	132014.3	127012.2	124291.6
		improvement					
		10 s.	**30 s.**	**60 s.**	**10 s.**	**30 s.**	**60 s.**
u_c_lohi.0		-3.6%	7.77%	**11.10%**	9.83%	11.87%	**12.56%**
u_i_hihi.0		5.9%	13.43%	**14.91%**	13.17%	15.17%	**15.97%**
u_s_lohi.0		-3.3%	7.25%	**10.03%**	5.90%	9.46%	**11.40%**

Table 6.29: Trade-off between solution quality and execution time (pCHC and pμ-CHC).

Table 6.29 shows that despite starting from worse solutions than the one computed by Min-Min, both pCHC and pμ-CHC are able to find well-suited schedules in a short time. Concerning pCHC, 10 s. are enough to achieve 6% of makespan improvement for inconsistent instances, while for consistent and semiconsistent instances, 30 s. are required to obtain more than 7% of improvement. When using 60 seconds of execution time, pCHC obtains an improvement of over 10% for all scenarios (and almost 15% for inconsistent instances). The pμ-CHC algorithm achieved significant makespan improvements (6% to 13%) over Min-Min using only 10 s. of wall-clock time, while larger improvements (11% to 16%) are obtained when using 60 s. of execution time. These results can be further enhanced by seeding the EA's populations with the full solution computed by the Min-Min or Sufferage deterministic heuristic.

The previous results demonstrate that both parallel models of the CHC algorithm proposed in this work are able to improve the deterministic heuristics results in a short execution time, a crucial result when scheduling in HC and grid computing systems.

6.7.2 Makespan evolution and execution time of pμ-CHC

This subsection extends the analysis of the makespan evolution for pμ-CHC, the best method among the studied EAs, considering the large-dimension HCSP instances designed in this work. Figure 6.16 presents the evolution of the averaged makespan improvement ratio achieved by pμ-CHC with respect to the best deterministic heuristic (i.e. the ratio between the pμ-CHC and the best deterministic heuristic makespan values), as a function of the execution time for each problem dimension studied.

The graph in Figure 6.16 shows that pμ-CHC was able to achieve accurate results in low execution times for all HCSP instances studied. Less than fifteen seconds are needed to achieve significant improvements with respect to the best deterministic heuristic in low-dimension problem instances (512×16 and 1024×32). The makespan reduction follows a slower behavior for the largest problem instances faced, but 30 s. are enough to achieve an improvement large than 5% in the makespan values, except for problems with dimension 8192×256 (where around a minute is needed to achieve 7% of makespan reduction).

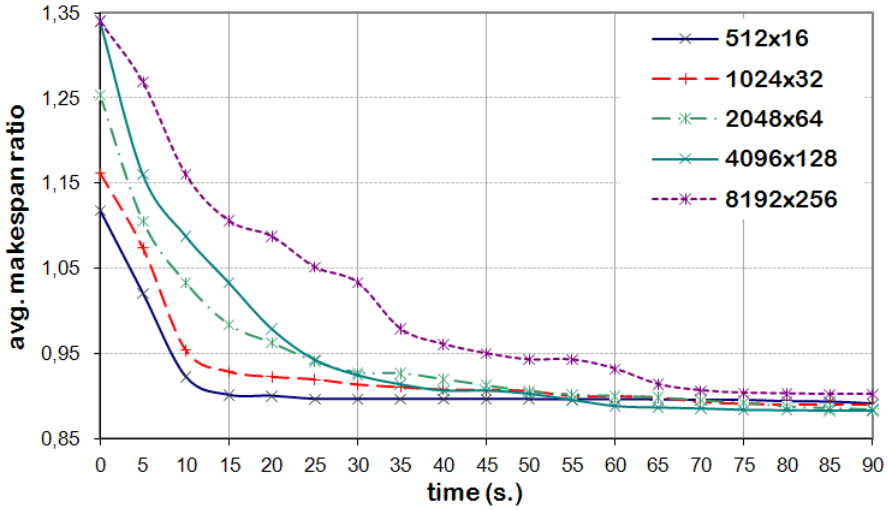


Figure 6.16: Evolution of the makespan improvement ratio for pμ-CHC.

From an execution time-oriented point of view, Figure 6.17 presents the time required to achieve a given improvement threshold in the makespan value with respect to the best deterministic heuristic result.

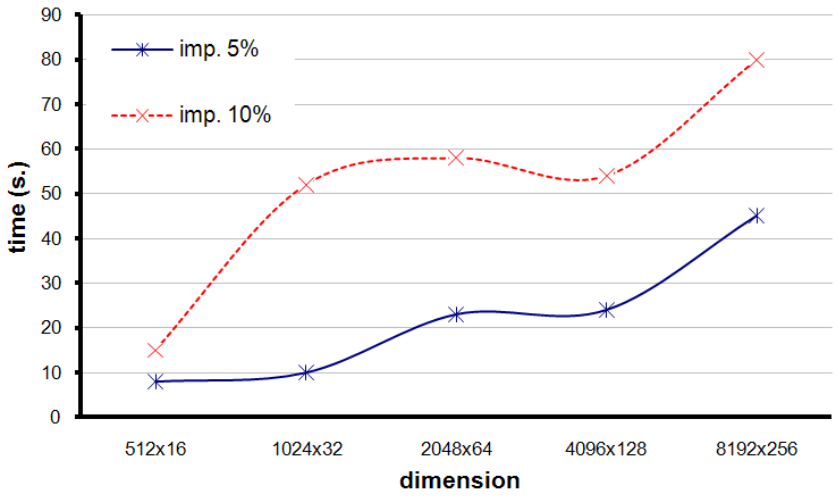


Figure 6.17: Execution times required to achieve a given improvement threshold.

The graphic in Figure 6.17 shows that pμ-CHC needs in average from 8 s. (for HCSP instances with dimension 512×16) to 45 s. (for HCSP instances with dimension 8192×256) of execution time to improve over the 5% threshold value. In addition, 15 s. (for HCSP instances with dimension 512×16) to 80 s. (for HCSP instances with dimension 8192×256) of execution time are required to achieve an improvement of 10% over the best deterministic result.

The previously commented results demonstrate the capacity of pμ-CHC to act as an efficient and accurate scheduler for HC and grid environments, able to improve over deterministic results in short execution times.

6.8 Scalability analysis and parallel performance

This section presents additional experiments performed to analyze the scalability and parallel performance of the proposed parallel CHC algorithms. The parameter setting experiments showed that the best makespan results for low-dimension HCSP instances were obtained using eight demes in pCHC and 16 demes in pμ-CHC. However, the parallel models could be able to achieve improved results by splitting the population in a different number of demes when facing the large HCSP instances.

The scalability analysis was aimed at studying the efficacy of pCHC and pμ-CHC as the HCSP instances dimension grows, and the parallel performance analysis was devoted to evaluate the results of pCHC and pμ-CHC when using different number of subpopulations to solve large-dimension problem instances. Instead of working with a fixed number of demes, the parallel CHC algorithms were executed using 2 to 16 subpopulations until reaching the time stopping criterion of 90 s. Mean values of the *normalized* makespan improvements (i.e. the quotient between the makespan achieved using p subpopulations and the makespan obtained with a single panmictic population) were evaluated for each problem instance, dimension, and consistency category. The results are presented and discussed in the next subsections.

6.8.1 Parallel CHC

Figure 6.18 presents the results in the scalability analysis that studied the behavior of pCHC when solving HCSP instances of increasing size. Mean values of the average normalized makespan achieved by pCHC using eight subpopulations in 30 independent executions of pμ-CHC are reported for consistent, inconsistent and semiconsistent instances for each problem dimension studied. The plot shows that the normalized makespan values diminish when solving large-dimension problem instances, specially for inconsistent and semiconsistent instances. Reductions of up to 10% in the mean makespan values were obtained when solving the largest problem instances (8192×256, semiconsistent instances), while the baseline results were around 3-5% for the smaller problem instances (512×16). These results demonstrate that the parallel model allows pCHC to have a good scalability behavior, able to achieve improved makespan reductions with respect to a panmictic model as the search space dimension grows.

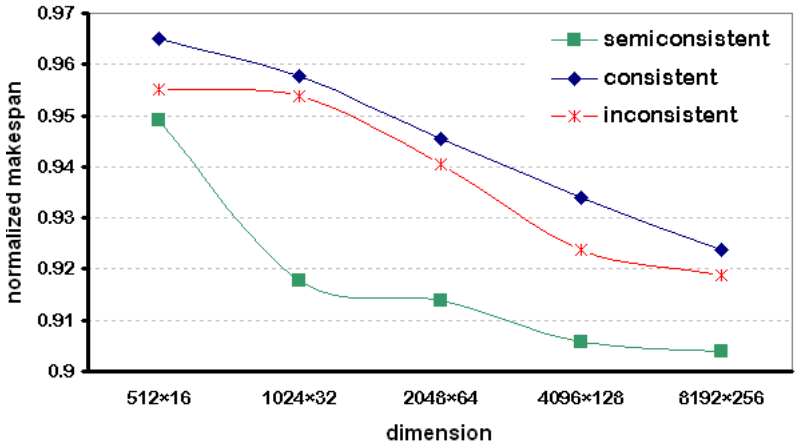
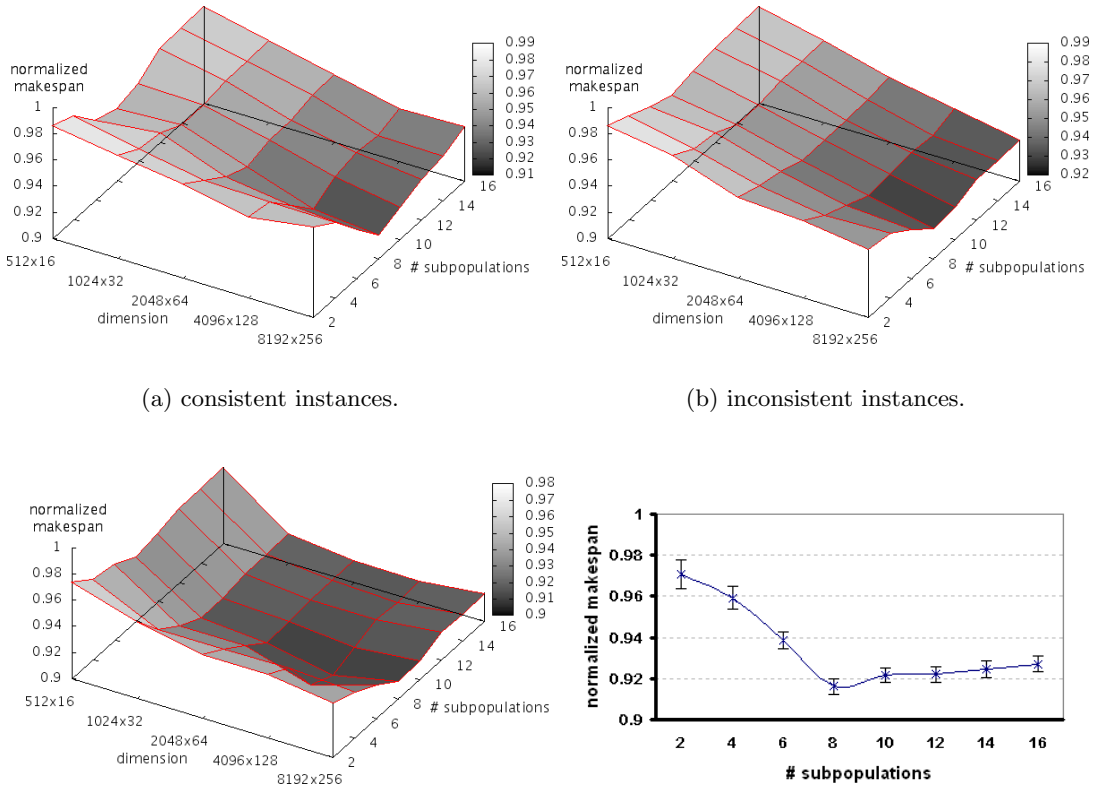


Figure 6.18: Scalability analysis for pCHC.

Figure 6.19 presents a three-dimensional graphical analysis of the parallel performance and scalability analysis of pCHC. By splitting the population, pCHC has a more focused exploration pattern than the panmictic search. However, the normalized makespan values show that splitting the population in more than eight demes causes the pCHC results to deteriorate, mainly due to the lose of diversity in each deme. A sample cut of the three-dimensional graphics for a representative dimension (2048×64, including error marks) is presented in Figure 6.19(d), showing the reduction of the normalized makespan values up to 8 subpopulations, and the slight worse results achieved when using additional subpopulations. Similar results were obtained when solving other HSCP instances. This behavior suggested that there was still work to be done to enhance the pCHC method in order to achieve a fully scalable scheduler, able to improve over its own results by using additional available computational resources. This idea led to the design of the pμ-CHC algorithm in a subsequent step of the project.



(a) consistent instances.

(b) inconsistent instances.

(c) semiconsistent instances.

(d) makespan variation (semiconsistent, 2048×64).

Figure 6.19: Parallel performance and scalability of pCHC.

6.8.2 Parallel micro CHC

Figure 6.20 reports the results in the scalability analysis that studied the behavior of pμ-CHC when solving HCSP instances of increasing size. Mean values of the average normalized makespan improvements achieved in 30 independent executions of pμ-CHC using 16 subpopulations are reported for consistent, inconsistent and semiconsistent instances for each problem dimension studied. The graphic shows that the normalized makespan values diminishes when solving large-dimension problem instances. Reductions of up to 10% in the mean makespan values were achieved when solving the largest problem instances (8192×256, semiconsistent instances), while the baseline results were around 5% for the smaller problem instances (512×16).

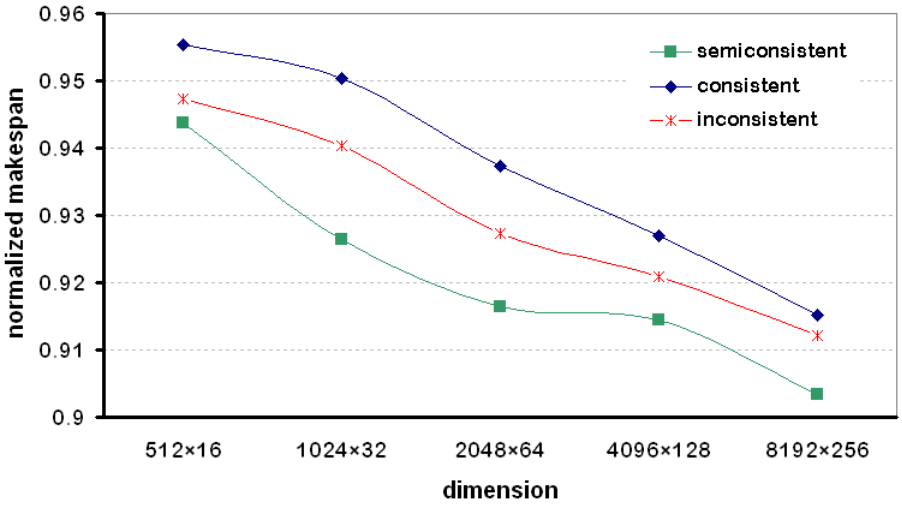
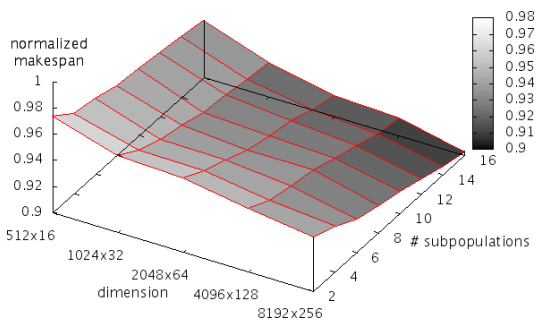
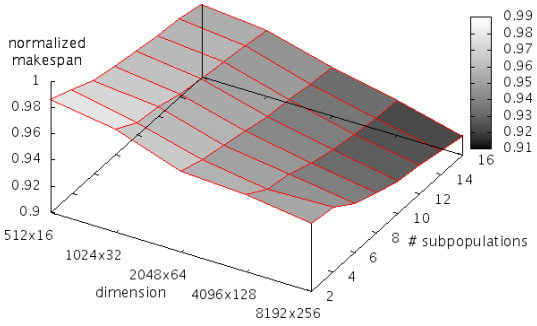


Figure 6.20: Scalability analysis for pμ-CHC.

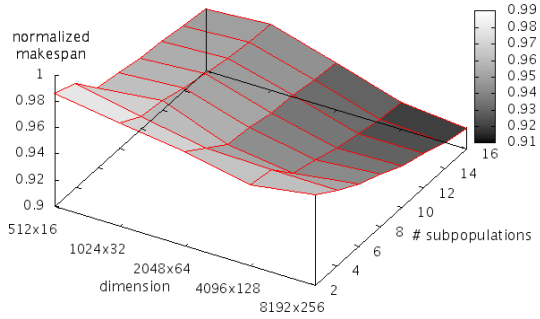
Figure 6.21 presents a 3D graphical analysis of the parallel performance and scalability analysis of pμ-CHC. The plots in Figure 6.21 show that pμ-CHC has a more focused exploration pattern than the panmictic search when using additional subpopulations. A sample cut of the 3D graphics for instances with a representative dimension (2048×64) and problem type (semiconsistent) is presented in Figure 6.21(d), showing the reduction of the normalized makespan values (including error marks). Similar results were achieved when solving other HSCP instances. The parallel performance of pCHC, already presented in the previous subsection, is included in Figure 6.21(d) in order to perform a comparison with the pμ-CHC behavior. The pμ-CHC algorithm always obtained the best results when using the maximum number of demes (16), demonstrating that by using the diversity provided by the randomized PALS operator, pμ-CHC overcomes the main problem of pCHC that suffered a degradation of the makespan results when more than eight subpopulations are used.



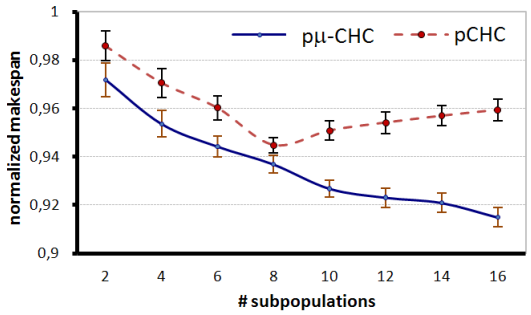
(a) consistent instances.



(b) inconsistent instances.



(c) semiconsistent instances.



(d) makespan variation sample (semiconsistent, 2048×64).

Figure 6.21: Scalability and parallel performance analysis for pμ-CHC.

The previous results demonstrate the ability of pμ-CHC of taking advantage of both the multiple evolutionary search and the randomized PALS operator in an efficient manner when using additional computational resources. The new pμ-CHC is a fully scalable scheduler, able to improve over its own results when using additional available computational resources.

6.9 Summary

This chapter presented an exhaustive experimental analysis on applying the proposed parallel EAs to solve a de-facto benchmark and new large-dimension instances of the HCSP. The analysis of the numerical results and the comparison with other techniques and lower bound allowed to conclude that the new pμ-CHC is the current state-of-art algorithm for scheduling in HCSP instances that follow the ETC performance estimation model by Ali et al. (2000). The pμ-CHC method improved over the previous best-known results for all the problem instances by Braun et al. (2001) and significantly improved over the results obtained with deterministic heuristics for the large-sized HCSP instances. The study of the scalability and parallel performance showed that pμ-CHC is able to obtain accurate schedules for all the studied HCSP instances in short execution times, while it is able to improve over its own makespan results when using additional computational resources, often available in HC and grid computing systems.

Chapter 7

Two HCSP variants: rescheduling and multiobjective approach

7.1 Introduction

This chapter presents the advances on using the new pμ-CHC algorithm and other parallel EAs to solve two HCSP variants: the scheduling problem that optimizes the makespan in a dynamic scenario where tasks arrive at certain intervals of time, and a multiobjective HCSP version that proposes the simultaneous optimization of the makespan and flowtime metrics. These two lines of research are proposed as future work to further investigate the applicability of parallel EAs for solving HCSP variants –more complex than the traditional one–, but some initial approaches have already been developed to tackle both problem versions. The main results of the preliminary analysis on both lines of research are presented in this chapter.

7.2 Dynamic HCSP: rescheduling with pμ-CHC

This section presents the application of the new pμ-CHC algorithm to the HCSP in dynamic scenarios following the rescheduling strategy, which performs several executions of the scheduling algorithm at certain intervals of time. The problem model and the main features of the rescheduling algorithms are described in the next subsections. After that, the following subsection presents the experimental analysis performed on three representative large-sized dynamic HCSP scenarios. Finally, brief conclusions about this preliminary analysis are formulated.

7.2.1 Dynamic HCSP model

The HCSP version faced in this section considers several bunches of tasks submitted to execution in a HC system. This is the typical scenario for scheduling in dynamic computing environments, where not all tasks are available a priori, but they arrive during the execution of previously submitted tasks.

In the problem model considered in this section, each bunch of tasks periodically arrives at certain intervals of time. Tasks are supposed to be submitted at the HC system according to a stochastic homogeneous Poisson process with rate λ (in the proposed scenarios $\lambda = 4 \times (NT/3) \times makespan$, where NT is the number of tasks in each bunch and $makespan$ is the computed makespan value for the previous bunch).

Each time that a new bunch of tasks arrives for execution, a *rescheduling* process is activated. The rescheduling strategy consists in finding a new schedule for executing the incoming tasks and also those tasks already submitted that have not been executed yet. Figure 7.1 graphically describes the process: in time T_R a reschedule is performed, and the new optimization problem considers the new tasks arrived plus all the tasks that have not started their execution at the time T_R, regarding the previously computed schedule. In the new scheduling problem, the calculation of the makespan metric must consider the remaining time of those tasks already in execution at time T_R in each machine.

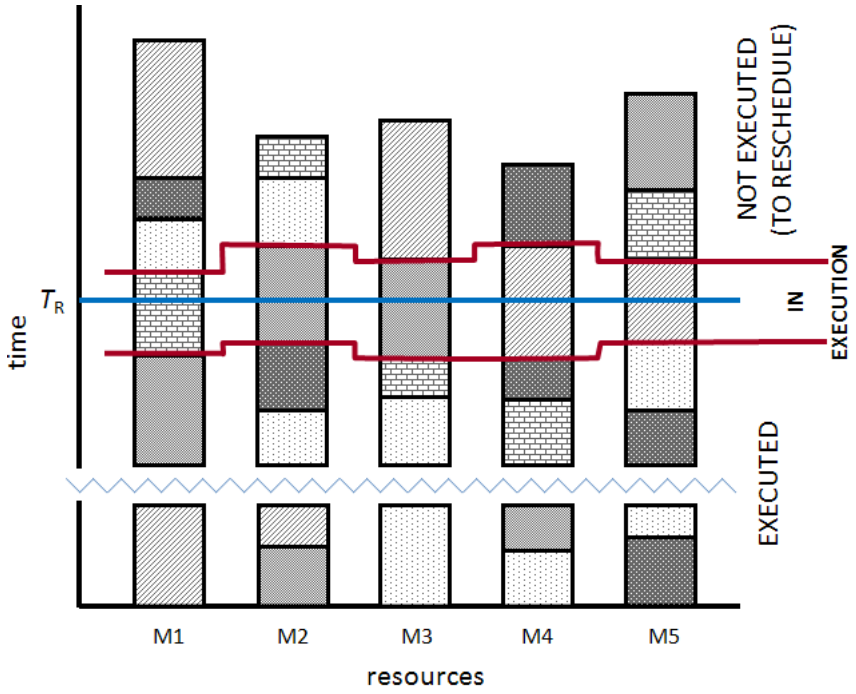


Figure 7.1: Rescheduling in a dynamic scenario.

By using this rescheduling strategy, the scheduler is able to handle dynamic environments in which tasks are supposed to arrive during the execution. Instead of solving several independent off-line scheduling problems for each bunch of tasks, performing a rescheduling at time T_R allows to take advantage of possible assignments of the remaining tasks to machines in order to improve the makespan metric of the whole schedule.

The rescheduling approach has been seldom studied in the related literature. One of the few antecedents in this line of research is the work by Theys et al. (2001), who proposed a methodology to perform online task rescheduling (in real time) using the off-line static mapping computed by a GA as a reference baseline. However, the approach by Theys et al. (2001) was not generic but corresponded to a problem-specific domain for a particular case of hardware platform.

7.2.2 Rescheduling algorithms

A specific pμ-CHC implementation was designed in order to deal with dynamic scenarios following the rescheduling strategy. The modifications performed over the traditional implementation of pμ-CHC –described in Chapter 5 included:

- a routine for reading ETC scenarios which is different to the one used for the traditional HCSP, since in the dynamic version the ETC matrix must consider the new tasks arrived in each bunch and those unexecuted tasks from the previous bunches;

- a modified fitness function that takes into account the local makespan values for each machine, considering the remaining time of those tasks already in execution at time T_R in each machine;

- a new function for storing the solutions –schedules– that must be read in the next rescheduling step in order to determine those tasks already executed, those tasks currently in execution at time T_R, and those tasks that have not been executed yet. All this information is required in order to perform the rescheduling using the new pμ-CHC method.

Two variants of the rescheduling pμ-CHC were studied: the first variant uses the seeded initialization described in Chapter 5 for the traditional HCSP, while the second variant employs an initialization operator that reuses the information already present in a previously computed schedule (i.e., the unexecuted tasks from the previous bunches are initially placed in those machines in which there were assigned in the best schedule computed in the last rescheduling step).

A modified version of the Min-Min deterministic heuristic was also implemented in order to evaluate the results obtained using the rescheduling pμ-CHC method. This Min-Min variant implements the rescheduling strategy in the same way that it was already described in the previous paragraphs for the pμ-CHC algorithm, by including the three features (reading routine, modified fitness function, and solution storing) needed to deal with dynamic scenarios.

In addition, a simple HC/grid simulator was designed and implemented. This software performs all the operations needed for simulating the scheduling and rescheduling in a dynamic HC/grid scenario, including the simulation of the tasks arrivals, the creation of ETC files for each bunch of tasks –using the generator program already introduced in Chapter 3–, the invocation of the scheduling/rescheduling method, and performing all the intermediate operations to communicate results from a previously computed schedule each time that a rescheduling is applied. The HC/grid simulator program follows the logic presented in the pseudocode in Algorithm 10. The Mersenne Twister pseudo-random number generator (Matsumoto and Nishimura, 1998) was used to implement the Poisson process for the arriving of tasks and the calculation of random times to perform the rescheduling. Mersenne Twister is a popular tool in the Monte Carlo simulation field, since it supplies high quality pseudorandom number sequences while also provides an efficient and portable implementation.

Algorithm 10 Pseudo-code of the HC/grid simulator program.

```
1: Configure simulator
2: Initialize pseudorandom number generator
3: Generate ETC matrices for all the bunches of tasks
4: for i = 1 to NUM_RESCHEDULING do
     {Rescheduling cycle}
5:   Configure execution
6:   if i > 1 then
7:     NT ← NT + NT_replan
8:     Add ETC of replanned tasks
9:   end if
     {Create child process to perform the schedule}
10:  if fork() == 0 then
11:    execlp(scheduler)
       {scheduler: pμ-CHC or Min-Min}
12:  end if
     {Wait for child process to execute}
13:  wait
14:  Process the computed schedule (read makespan, tasks, etc.)
15:  NT_replan ← 0
     {Compute new rescheduling time (TR)}
16:  TR ← TR = 0.5× makespan × (1+genrand64_real1());
17:  for each machine M do
18:    for each unexecuted task in time TR do
         {Add task to the list of replanned tasks}
19:      NT_replan ← NT_replan + 1
20:    end for
21:  end for
22: end for
```

The HC/grid simulator described by the pseudocode in Algorithm 10 performs a rescheduling cycle (from 1 to NUM_RESCHEDULING scheduling steps). Except for the first bunch of tasks (when no previous tasks exist), the number of tasks to schedule in the current iteration is the sum of the NT arriving tasks plus the NT_replan replanned tasks in the previous iteration. In each rescheduling step, the HC/grid simulator configures and creates a child process to execute the scheduling algorithm (either pμ-CHC or Min-Min, in their correspondent variants adapted to apply the rescheduling strategy). The next rescheduling time TR is randomly chosen in the interval [*makespan*/2,*makespan*], where *makespan* is the value of the makespan metric computed by the execution of the correspondent scheduling method for the previous bunch of tasks. After that, the previous best schedule is analyzed, and for each machine, each unexecuted task is included in the list of replanned tasks, in order to be considered in the next rescheduling iteration.

7.2.3 Experimental analysis

This subsection presents the experimental evaluation of the pμ-CHC method for solving the dynamic version of the HCSP using the rescheduling strategy. Three representative dynamic scenarios with large dimension are studied, and the Min-Min results are included as a reference baseline for the comparison.

HCSP instances

The experimental evaluation was performed using a set of three HCSP instances: a medium-size instance with 25 machines and 10000 tasks (arriving in 10 bunches of 1000 tasks), and two large HCSP instances: one with 50 machines and 25000 tasks (arriving in 25 bunches of 1000 tasks), and the other with 100 machines and 50000 tasks (arriving in 50 bunches of 1000 tasks). All the problem instances were generated using the HCSP generator program introduced in Chapter 3, following the parametrization model from Braun et al. (2001) for high task and machine heterogeneity, since this model represents the most generic scenario for scheduling in HC systems. The HCSP instances are labeled with a name that indicates the number of tasks and machines involved in the scenario (i.e. the medium-size HCSP instance named `10000_25` represents a HC scheduling scenario with 10000 tasks and 25 machines).

Development and execution platform

The standard pμ-CHC implementation already developed in MALLBA was used as a skeleton to implement the modifications needed to perform the rescheduling strategy. The HC/grid simulator was implemented in C++, including the `mt19937-64.c` C/C++ implementation of the Mersenne Twister pseudorandom number generator, a variant that uses a 64-bit word length for defining the period of the pseudorandom sequence. The code of `mt19937-64.c` is publicly available at the Hiroshima University website `http://www.math.sci.hiroshima-u.ac.jp/ m-mat/MT/VERSIONS/C-LANG/mt19937-64.c` (Nishimura, 2000).

The experimental analysis was performed using the cluster of four Dell PowerEdge servers in our workgroup, whose details were already described in Chapter 6 (cluster website: `http://www.fing.edu.uy/cluster`).

Experimental results

Table 7.1 presents the best and average makespan results obtained in 10 independent executions of the HC/grid simulator using the pμ-CHC and Min-Min algorithms following the rescheduling strategy for each studied scenario. The results obtained using the traditional pμ-CHC and Min-Min (without rescheduling) are also presented as a reference baseline for the comparison. In both the rescheduling and traditional variants, pμ-CHC was executed using a stopping criterion of 60 s. for scheduling/rescheduling each bunch of tasks.

Instance	**pμ-CHC** (rescheduling)		**pμ-CHC** (traditional)		**Min-Min** (rescheduling)	**Min-Min** (traditional)
	best	avg.	best	avg.		
10000_25	**22785530.0**	22894922.5	23138767.9	23172507.6	28275400.0	28902847.5
25000_50	**15747029.1**	16030865.0	16119560.1	16223856.7	21154820.2	22416536.4
50000_100	**9619025.0**	10088298.9	10450847.0	10676603.5	12519651.4	15651766.5

Table 7.1: Makespan results of pμ-CHC and Min-Min for the dynamic HCSP.

The results reported in Table 7.1 correspond to the rescheduling pμ-CHC algorithm using the seeded initialization described in Chapter 5 for the single-objective HCSP. No significant improvements were detected in the makespan results obtained by the other studied variant of pμ-CHC –using the initialization operator that reuses the information already present in a previously computed schedule–, so it was omitted in the comparative analysis of makespan results.

The analysis of Table 7.1 show that the pμ-CHC algorithm using the rescheduling strategy significantly improves over the makespan results computed with the other scheduling methods studied (traditional Min-Min, Min-Min with rescheduling and traditional pμ-CHC). The largest makespan improvements were obtained for those scenarios involving the highest number of tasks. Table 7.2 summarizes the makespan improvements when using the rescheduling strategy in pμ-CHC with respect to the other scheduling methods.

Instance		**Min-Min** (traditional)	**Min-Min** (rescheduling)	**pμ-CHC** (traditional)
10000_25	best	21.17%	19.42%	1.53%
	avg.	20.79%	19.03%	1.20%
25000_50	best	29.75%	25.56%	2.31%
	avg.	28.49%	24.22%	1.19%
50000_100	best	38.54%	23.17%	7,96%
	avg.	35.55%	19.42%	5.51%

Table 7.2: Improvements of pμ-CHC using the rescheduling strategy over the traditional pμ-CHC and the two Min-Min variants.

The analysis of Table 7.2 shows that the makespan improvements obtained with the rescheduling pμ-CHC over the traditional pμ-CHC and the two Min-Min variants significantly increase when a large number of tasks are submitted to execution. Up to **7.96%** of improvement in the makespan values over the traditional pμ-CHC was achieved for the 50000_100 HCSP instance (5.51% in average). The improvements over the Min-Min schedulers were above **19%** for the three dynamic problem instances studied. The previous results suggest that pμ-CHC is able to efficiently handle dynamic HC scenarios by using the rescheduling strategy.

7.2.4 Concluding remarks

The preliminary study presented in this section analyzed the ability of the pμ-CHC algorithm using a rescheduling strategy to deal with dynamic HCSP scenarios. The reported results show that significant makespan improvements can be obtained when using the rescheduling pμ-CHC method, specially in those HCSP scenarios where a large number of tasks are submitted to execution. Further work and experimental analysis are needed in order to clearly determine the benefits of the rescheduling strategy and comparing the rescheduling pμ-CHC algorithm with other dynamic and on-line methods for scheduling.

7.3 Multiobjective HCSP: makespan and flowtime

This section presents the application of the new pμ-CHC algorithm and other parallel EAs to the multiobjective HCSP variant aimed at simultaneously optimizing the makespan and flowtime metrics. Next sections provide a brief introduction to multiobjective optimization problems, just before presenting the multiobjective HCSP formulation. After that, generic concepts about multiobjective evolutionary algorithms are introduced, and the specific parallel MOEAS for the HCSP are described. The following subsection presents the details of the experimental analysis, mainly focused in studying several metrics of solutions quality and diversity. Finally, brief conclusions of the preliminary research are provided and the main lines for future work are commented.

7.3.1 Introduction

Most traditional optimization problems tackled in the past considered only one objective function. Taking into account the multiobjective nature of most real world problems, in the last fifteen years more attention has been focused on multiobjective problems, which propose the optimization of a *vector function* representing several objectives, usually in conflict with each other. In this context, *optimizing* means finding solutions that represent all combinations of the objective function values that are acceptable for the problem. There is not a unique "optimum" solution as in single-objective optimization, and the decision about determining the "best" results corresponds to the –often human, rarely automatized– *decision making procedure*, applied after the search. From a global point of view, there exists a set of "optimal" solutions (called *Pareto optimal solutions*), which represent different trade-offs among the objective values (Coello et al., 2002).

EAs have been successfully applied to solve problems requiring the simultaneous optimization of many objectives. By taking advantage of working with a set of possible solutions in each generation, multiobjective evolutionary algorithms (MOEAs) perform a robust search, frequently finding several members of the Pareto optimal set in a single execution, and being able to explore intractably large spaces of complex problems (Deb, 2001; Coello et al., 2002).

7.3.2 Multiobjective optimization problems

This subsection presents a brief introduction to multiobjective optimization problems and related concepts.

A multiobjective optimization problem (MOP) proposes the optimization (minimization or maximization) of a group of functions, usually in conflict with each other. The existence of several objective functions implies a fundamental difference with a single-optimization problem: there is not a unique solution for the optimization problem, but a set of solutions that represent several trade-offs among the values of the functions to optimize. The general formulation of a MOP is presented in the following lines:

$$\begin{aligned} \text{minimize/maximize} \quad & \mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})) \\ \text{subject to} \quad & \mathbf{G}(\mathbf{x}) = (g_1(\mathbf{x}), g_2(\mathbf{x}), \ldots, g_j(\mathbf{x})) \geq 0 \\ & \mathbf{H}(\mathbf{x}) = (h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_l(\mathbf{x})) = 0 \\ & {x_i}^{(L)} \leq x_i \leq {x_i}^{(U)} \end{aligned}$$

The solution of the MOP is a vector of decision variables $\mathbf{x} = (x_1, x_2, \ldots, x_N) \in \Omega$ which satisfies the constraints formulated by the functions $\mathbf{G}(\mathbf{x})$ and $\mathbf{H}(\mathbf{x})$, offering adequate compromise values for the functions $f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})$. Let's suppose a minimization MOP for introducing the following definitions. A solution $\mathbf{x}^*$ is *Pareto optimal* if for all $\mathbf{x}$, $f_i(\mathbf{x}) \geq f_i(\mathbf{x}^*)$ and for some i, $f_i(\mathbf{x}) > f_i(\mathbf{x}^*)$. This condition means that there not exists a feasible solution that is "better" than $\mathbf{x}^*$ in any objective function without being "worse" in other functions.

Associated with the precedent definition, a partial order relation named *dominance* is introduced among solutions of the MOP. A vector $\mathbf{w} = (w_1, w_2, \ldots, w_N)$ is said to dominate the other solution $\mathbf{v} = (v_1, v_2, \ldots, v_N)$ (it is denoted $w \prec v$), if $f_i(\mathbf{x}) \geq f_i(\mathbf{w})$ and it exists at least one objective function f_j such as $f_j(\mathbf{x}) > f_j(\mathbf{w})$.

Since different values of the decision variables represent diverse compromise among the solutions, the resolution of a MOP does not focus on finding a unique solution, but a set of non-dominated solutions. The set of optimal solutions for a MOP is composed by the non-dominated feasible vectors, named *Pareto optimal set*. It is defined by $P^* = \{\mathbf{x} \in \Omega / \nexists \mathbf{x}' \in \Omega, \mathbf{F}(\mathbf{x}') \preceq \mathbf{F}(\mathbf{x})\}$. The region of points defined by the optimal Pareto set in the objective function space is known as *Pareto front*, formally defined by $PF^* = \{\mathbf{u} = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})), \mathbf{x} \in P^*\}$.

7.3.3 Multiobjective HCSP formulation

The multiobjective formulation of the HCSP faced in this work considers the minimization of the *makespan* and the *flowtime* metrics. The makespan was already defined in Chapter 3, while the flowtime evaluates the sum of the finishing times of tasks executed in a HC system. The flowtime is an important metric from the user point-of-view, since it reflects the response (in time) of the computational system for a set of tasks submitted to execution by the users.

Based on problem model introduced in Chapter 3, the following formulation defines the multiobjective version of the HCSP aimed at simultaneously minimizing the makespan and flowtime:

- Given an HC system composed of a set of computational resources (machines) $P = \{m_1, m_2, \ldots, m_M\}$ (dimension M), and a collection of tasks $T = \{t_1, t_2, \ldots, t_N\}$ (dimension N) to be executed on the HC system;
- let there be an *execution time function* $ET : P \times T \rightarrow \mathbf{R}^+$, where $ET(t_i, m_j)$ is the time required to execute the task t_i in the machine m_j;
- let $F(t_i)$ be the finishing time of task t_i in the HC system;
- the goal of the multiobjective HCSP is to find a set of assignment of tasks to machines which simultaneously minimizes the *makespan* –defined in Equation 7.1– and the *flowtime* –defined in Equation 7.2–.

$$\max_{m_j \in P} \sum_{\substack{t_i \in T: \\ f(t_i) = m_j}} ET(t_i, m_j) \tag{7.1}$$

$$\sum_{t_i \in T} F(t_i) \tag{7.2}$$

Computing the makespan metric does not need sorting the set of tasks to be executed in every machine, but the flowtime calculation assumes an explicit order: the assigned tasks are supposed to be executed in ascending order of their ETC value, since this order minimizes the finishing times of the tasks. While makespan and flowtime are somehow related metrics, they are also conflicting objectives to minimize, specially for near-optimal schedules (Xhafa et al., 2008b).

The presented multiobjective HCSP version has been previously solved with evolutionary techniques in the works by Xhafa et al. (2008a) using a cellular MA, and by Xhafa (2007) using a hybrid method combining MA and TS. However, both works used single objective methods using a linear aggregation approach to deal with an explicit multiobjective problem. There have not been precedents in the related literature about solving the makespan-flowtime HCSP variant using explicit multiobjective algorithms, thus the proposed MOEAs are a direct contribution in this line of research.

7.3.4 Multiobjective evolutionary algorithms

Back in 1967, Rosenberg hinted at the capability of evolutionary computation techniques for solving multiobjective optimization problems. However, the first algorithmic proposal for a MOEA was not presented until a pioneering work by Schaffer in 1984. Since 1990, multiple MOEAs have been proposed by a growing research community that works actively in the present days. In those works, EAs have been successfully extended to deal with multiobjective problems, showing promising results in multiple application areas (Deb, 2001; Coello et al., 2002).

Unlike many traditional methods for multiobjective optimization, EAs are able to find a set of multiobjective solutions in a single execution, since they work with a population of tentative solutions in each generation. MOEAs have to be designed having in mind two goals at the same time: to approximate the Pareto front and to maintain the diversity among solutions, obtaining a good sampling of the whole Pareto front instead of converging to a single solution or a reduced section. A modified version of the evolutionary search leads to the first goal, while the second one is accomplished by using specific techniques also employed in multimodal function optimization (niches, sharing, crowding, etc.).

The multiobjective HCSP version faced in this section was solved using four different parallel MOEAs: two implementations of a distributed subpopulations parallel version of NSGA-II, a master-slave parallel version of SPEA-2 and the new pμ-CHC algorithm proposed in this thesis. The approach that proposes using explicit multiobjective algorithms –instead of applying single-objective EAs adapted to work with a linear aggregation fitness function– to solve the makespan-flowtime optimization version of the HCSP is a novel contribution of this work. The details of the proposed MOEAs are briefly presented in the following subsections.

NSGA-II multiobjective evolutionary algorithm

Non-dominated Sorting Genetic Algorithm, version II (NSGA-II) (Deb et al., 2000) surged as an improved version of its predecessor NSGA (Srinivas and Deb, 1994). NSGA-II inherited the main structure of NSGA, but it incorporated distinctive features to deal with tree algorithmic issues strongly criticized on the previous version: the inefficient non-dominated ordering, the lack of elitism, and the σ_{SH} parameter dependence.

The main features of NSGA-II include:

- the non-dominated elitist ordering using an auxiliary subpopulation, diminishing the dominance checking complexity from $O(M.P^3)$ to $O(M.P^2)$, being M the number of objective functions and P the population size;
- the mechanism for diversity preservation eliminates the σ_{SH} dependence by using a crowding technique that does not demand to specify additional parameters.
- the fitness assignment procedure, still based on non-dominance ranks but incorporating the crowding distance values which are used to evaluate the diversity of the solutions.

The multiobjective HCSP was solved using two parallel versions of NSGA-II following the distributed subpopulations model for EAs (Alba et al., 2002): pNSGA-II, a parallel NSGA-II implementation designed by the author (Nesmachnow, 2004), and MOE-NSGA-II an implementation included in the MOE framework (Rodríguez and Nesmachnow, 2009). Both parallel implementations use MPI for the interprocess communication and synchronization. pNSGA-II uses an explicit send/receive schema for communication, while MOE-NSGA-II employs a more sophisticated protocol similar to the one implemented in the `Netstream` class in the MALLBA library.

The pNSGA-II algorithm was previously applied with success to solve twelve well-known standard multiobjective test problems, showing super-linear speedup behavior and also finding accurate solutions for a hard multiobjective network design problem (Nesmachnow, 2005). The parallel NSGA-II implementation in the MOE framework was also tested for a set of standard multiobjective problems, obtaining accurate numerical and efficiency results (Rodríguez and Nesmachnow, 2009).

SPEA-2 multiobjective evolutionary algorithm

Strength Pareto Evolutionary Algorithm, version 2 (SPEA-2) was introduced by Zitzler et al. (2001) to eliminate the potential weaknesses of the original SPEA algorithm (Zitzler and Thiele, 1999) in order to design a powerful and up-to-date MOEA. The main differences of SPEA-2 in comparison to SPEA are:

- SPEA-2 includes an improved fitness assignment scheme, taking for each individual into account how many individuals it dominates and it is dominated by;
- a nearest neighbor density estimation technique was incorporated in SPEA-2, which allows a more precise guidance of the search process;
- SPEA-2 uses an improved archive truncation method that guarantees the preservation of boundary solutions in the elite population.

The SPEA-2 algorithm used in this work is implemented in `jMetal` (Durillo et al., 2006), a software framework aimed at making easy to develop metaheuristic algorithms for solving MOPs. `jMetal` does not provide support for the execution of parallel MOEAs. The parallel version of SPEA-2 (pSPEA-2) was implemented using the `sparrow` library (Durillo et al., 2008), which allows defining and executing concurrent tasks. A master-slave PEA model was used in the parallel version of SPEA-2 –mainly due to the simplicity of the model– using the three components available in `sparrow`: `worker`, `driver`, and `workerServer`. The pSPEA-2 algorithm follows the traditional master-slave approach that distributes the fitness function evaluation, already described in Chapter 2.

pμ-CHC

The pμ-CHC algorithm was not initially conceived to solve multiobjective optimization problems. The random PALS operator was used in pμ-CHC to implicitly improve the population diversity, without using an explicit method for sampling Pareto-optimal solutions. In addition, pμ-CHC does not use a Pareto-based schema for assigning fitness to solutions, then the algorithm does not guarantee –a-priori– a good sampling of the Pareto front for multiobjective optimization problems.

In order to study the applicability of pμ-CHC to solve the multiobjective HCSP version aimed at minimizing makespan and flowtime, the research reported in this section uses a linear aggregation function for fitness assignment, following the approaches previously used by Xhafa et al. (2007b, 2008a,b).

Many shortfalls and drawbacks have been identified for the linear combination technique for fitness assignment in multiobjective optimization (Das and Dennis, 1997; Fonseca and Fleming, 1997). However, the linear aggregation is still a popular approach due to its simplicity, which often allows finding accurate compromise solutions for problems with convex Pareto fronts.

In the pμ-CHC algorithm, the makespan and flowtime objective functions are combined using the weights already employed in previous works by Xhafa et al. (2007b, 2008a,b). The aggregation fitness function is given by the expression in Equation 7.3.

$$0.75 \times makespan + 0.25 \times flowtime \tag{7.3}$$

This decision allows performing a fair comparison of the pμ-CHC results with those obtained by Xhafa et al. (2007b, 2008a,b) for the set of HCSP instances by Braun et al. (2001).

7.3.5 MOEAs for the HCSP

The implemented MOEAs for the HCSP follow the generic proposal used in the single objective GA and CHC presented in Chapter 5. The machine-oriented encoding was employed in all MOEAs, since it provides a simple and efficient way to compute the makespan and flowtime metrics. pNSGA-II, MOE-NSGA-II, and pSPEA-2 use the uniform crossover and the mutation/local search already employed in the single objective pCHC implementation. pNSGA-II and pSPEA-2 follow a traditional evolution model, applying the mutation operator with probability p_M. The accelerated evolution model proposed for the pμ-CHC algorithm was adopted in MOE-NSGA-II, which applies the mutation/local search operator after five generations pass, disregarding convergence issues.

No special parameter setting experiments were performed to determine the best values for population size, and crossover and mutation/local search probabilities (p_C and p_M, respectively). Instead, probability values derived from the parameter settings experiments for the single-objective EAs were employed (population size = 120 individuals, $p_C = 0.7$, $p_M = 0.1$). The migration in the distributed subpopulations parallel MOEAs also used the parametrization from the single objective pμ-CHC algorithm: the migration operator exchanges the best two individuals between subpopulations (elitist selection), considering the demes connected in a unidirectional ring topology. The migration frequency was set at 500 generations, trying to achieve a balance between providing diversity and reducing the time spent in communications.

Regarding the number of subpopulations, pNSGA-II and MOE-NSGA-II worked using eight demes (this value was derived from preliminary experiments on configuring the parallel MOEAs), and pμ-CHC used 16 subpopulations. A unique panmictic population was used in pSPEA-2, since this algorithm adopted the master-slave parallelization approach.

7.3.6 Experimental analysis

This subsection presents the experimental evaluation of the proposed parallel MOEAs for solving the multiobjective version of the HCSP.

HCSP instances

The experimental evaluation was performed using the set of small-sized de-facto standard benchmark HCSP instances from Braun et al. (2001). Some preliminary experiments have been performed using the set of large-sized HCSP instances with pNSGA-II and pμ-CHC, but the results are still not formalized, thus they are not presented in this thesis.

Development and execution platform

The pNSGA-II algorithm was implemented in C, using the publicly available code from the Kanpur Genetic Algorithms Laboratory (website `http://www.iitk.ac.in/kangal`) (Nesmachnow, 2004). MOE-NSGA-II was implemented in C++, using and extending some classes from the MALLBA library (Rodríguez and Nesmachnow, 2009). pμ-CHC was also implemented in C++, using the MALLBA library. All the previously mentioned parallel MOEAs use MPICH for performing the interprocess communication. pSPEA-2 is implemented in Java, using the `jMetal` framework and the `sparrow` library for the definition and the execution of concurrent tasks, and also for performing the interprocess communication.

The experimental analysis was performed using the cluster of four Dell PowerEdge servers in our workgroup, whose details were already described in Chapter 6 (cluster website: `http://www.fing.edu.uy/cluster`).

Performance metrics

Many metrics have been proposed in the related literature in order to evaluate the results obtained by MOEAs (Coello et al., 2002). Several relevant metrics have been considered in this work to evaluate the proposed parallel MOEAs, regarding the purposes of converging to the Pareto front and sampling the set of non-dominated solutions:

- The number of (different) non-dominated solutions found (denoted by *ND*).
- The number of optimal Pareto solutions found (#P).
- Generational Distance (GD): the average distance among the non-dominated solutions found and the true Pareto front, as defined in Equation 7.4.

$$GD = \frac{1}{ND} \left(\sum_{i=1}^{ND} \left(d_i^{FP} \right)^k \right)^{\frac{1}{k}} \tag{7.4}$$

- Spacing: a metric that evaluates the distribution of non-dominated solutions in the calculated Pareto front, defined by the expression given in Equation 7.5.

$$spacing = \sqrt{\frac{1}{ND-1}\sum_{i=1}^{ND}\left(\bar{d}-d_i\right)^2} \tag{7.5}$$

- Spread: a metric that evaluates the distribution of non-dominated solutions in the calculated Pareto front. Unlike spacing, the spread metric includes the information about the extreme points of the true Pareto front in order to compute a most precise value of the distribution (Deb, 2001). The spread metric is defined by the expression given in Equation 7.6.

$$spread = \frac{\sum_{h=1}^{k} {d_h}^e + \sum_{i=1}^{ND}\left(\bar{d}-d_i\right)^2}{\sum_{h=1}^{k} {d_h}^e + ND \times \bar{d}} \tag{7.6}$$

In Equation 7.4, d_i^{FP} stands for the distance in the objective functions space between the i-th solution in the calculated Pareto front and the nearest point in the true Pareto front. In Equations 7.5 and 7.6 d_i denotes the distance between the i-th solution in the calculated Pareto front and its nearest neighbor (the j-th solution), while $\bar{d}$ is the average distance value of all d_i. In Equation 7.6 ${d_h}^e$ is the distance between the extreme point of the true Pareto front, considering the h-th objective function and the closest point in the calculated Pareto front.

The *efficacy* metrics (ND, #P, and GD) evaluate the quality of the results obtained using a MOEA, by computing measures related to the convergence towards the Pareto front. On the other hand, the *diversity* metrics (spacing and spread) evaluate the distribution of the computed non-dominated solutions, measuring the capability of sampling the Pareto front of the problem.

Some of the presented metrics require to know the true Pareto front for the problem, which is unknown for HCSP the instances studied. In order to compute those metrics, an approximation of the true Pareto front was built gathering the non-dominated solutions obtained using the four EAs in the 30 independent executions performed for each algorithm.

Results and discussion

Table 7.3 presents the best makespan and flowtime results obtained in 30 independent executions of the four parallel MOEAs studied for solving the HCSP instances by Braun et al. (2001). The best makespan and flowtime results computed for each problem instance –which correspond with the extreme values for each objective function– are marked with bold font.

Instance	pNSGA-II		pSPEA-2		MOE-NSGA-II		pμ-CHC	
	makespan	flowtime	makespan	flowtime	makespan	flowtime	makespan	flowtime
u_c_hihi.0	2968900.0	1035113210.0	7664708.5	1035559104.0	7669870.0	1034100000.0	**7595027.5**	**1033251173.4**
u_c_hilo.0	78033.8	12430900.0	154800.4	27511050.0	154650.0	**27470600.0**	**153997.5**	27494220.4
u_c_lohi.0	253509.0	34441600.0	250619.2	34406000.0	250148.0	**34319000.0**	**246392.7**	34340362.5
u_c_lolo.0	5245.7	915685.0	5239.0	915994.4	5230.9	914269.0	5187.4	**913252.5**
u_i_hihi.0	2968900.0	350100992.0	2972172.0	350388640.0	**2964590.0**	**350056000.0**	2966065.8	350190000.5
u_i_hilo.0	73780.6	**12430900.0**	74043.5	12434667.0	73825.1	12433400.0	**73674.4**	12433973.2
u_i_lohi.0	103286.0	12240400.0	103379.6	**12228552.0**	103336.0	12230400.0	**102732.6**	12231807.1
u_i_lolo.0	2562.5	434221.0	2567.3	434309.0	**2561.5**	**434148.0**	2564.7	434524.3
u_s_hihi.0	4303110.0	3497.5	4313499.0	505288224.0	4273710.0	**504447000.0**	**4201963.5**	505027968.6
u_s_hilo.0	96874.4	16194100.0	97138.2	16194711.0	96855.9	**16181900.0**	**96649.5**	16187925.1
u_s_lohi.0	125140.0	14858600.0	125533.7	14863620.0	125121.0	14848700.0	**124650.2**	**14843952.2**
u_s_lolo.0	3497.5	591431.0	3497.7	591651.1	3490.2	591381.0	**3482.1**	**591169.6**

Table 7.3: Results of MOEAs for the multiobjective HCSP.

The analysis of Table 7.3 shows that MOE-NSGA-II and pμ-CHC obtained the best values of makespan and flowtime for the majority of the HCSP instances faced. pNSGA-II and pSPEA-2 only obtained the best flowtime solutions in one inconsistent HCSP instance each. These results demonstrate the usefulness of the accelerated evolution model employed in MOE-NSGA-II and pμ-CHC to achieve accurate solutions in short execution times, confirming previous results obtained for the single-objective HCSP.

pμ-CHC consistently obtained the best makespan values, mainly due to the makespan-oriented local search provided by the randomized PALS method, but its best flowtime results were not as good as the makespan ones. MOE-NSGA-II found solutions with accurate values for the flowtime metric and reasonable values of makespan, suggesting that the Pareto-based fitness assignment technique is useful for obtaining solutions with better trade-off values between the two objectives than the linear aggregation model used in pμ-CHC.

Table 7.4 presents the average number of non-dominated solutions, the average number of Pareto-optimal solutions and the average values of the generational distance metric, all computed in the 30 independent executions for each parallel MOEA.

Instance	pNSGA-II			pSPEA-2			MOE-NSGA-II			pμ-CHC		
	ND	#P	GD	ND	#P	GD	ND	#P	GD	ND	#P	GD
u_c_hihi.0	**39.1**	0.0	1561423.0	20.0	0.0	99094.2	20.9	0.0	405480.5	6.2	**6.0**	**0.0**
u_c_hilo.0	**29.0**	0.0	12434.1	18.5	0.0	15961.1	21.6	**11.9**	3709.3	6.4	4.7	**199.0**
u_c_lohi.0	17.8	0.0	18990.3	20.7	0.0	17120.8	**28.2**	4.0	7682.1	8.0	**8.0**	**0.0**
u_c_lolo.0	26.2	0.0	1256.9	**30.1**	0.0	698.8	16.7	0.0	333.5	6.7	**5.8**	**0.0**
u_i_hihi.0	**38.1**	2.1	341418.4	29.2	2.0	98615.1	36.0	**19.5**	67952.4	9.7	3.3	**39082.9**
u_i_hilo.0	**27.7**	**10.7**	4372.1	27.4	0.0	7694.1	15.6	0.0	4152.8	8.4	5.6	**2039.6**
u_i_lohi.0	34.1	0.0	5993.0	**35.0**	**14.7**	2272.1	32.3	5.7	2709.5	9.1	8.2	**0.0**
u_i_lolo.0	**23.0**	4.0	416.7	15.1	0.5	42.7	20.2	**13.3**	**7.5**	8.7	0.0	856.5
u_s_hihi.0	**37.4**	0.0	248024.0	33.30	0.0	267548.7	31.4	5.8	215693.3	12.8	**9.6**	**30321.9**
u_s_hilo.0	20.9	**12.9**	**119.1**	17.8	0.7	3735.3	**28.5**	10.9	1072.3	9.1	1.3	1483.5
u_s_lohi.0	**39.5**	**10.7**	5921.3	29.9	1.2	5437.1	29.1	9.4	3958.8	8.5	5.1	**977.2**
u_s_lolo.0	25.1	3.0	571.5	27.2	0.0	138.9	**31.6**	3.8	199.2	7.0	**6.3**	**0.0**

Table 7.4: Efficacy metrics of MOEAs for the multiobjective HCSP.

Table 7.5 presents the average values of the spread and spacing metrics computed in the 30 independent executions of each parallel MOEA.

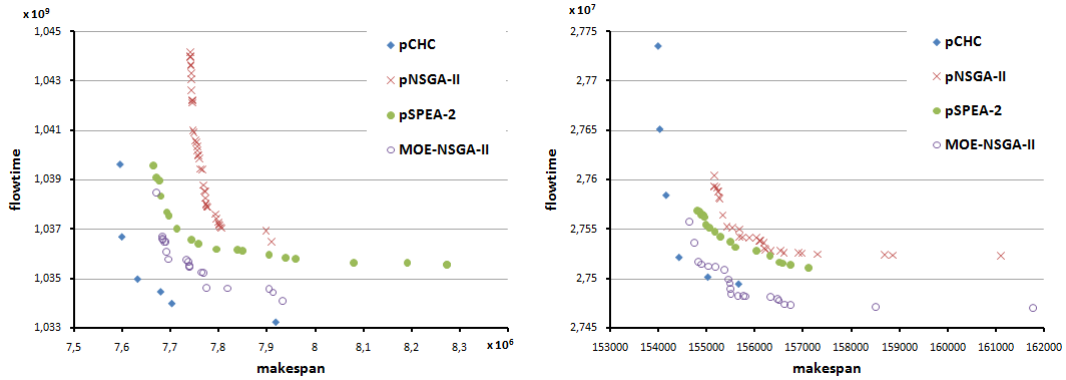
Instance	pNSGA-II		pSPEA-2		MOE-NSGA-II		pμ-CHC	
	spacing	spread	spacing	spread	spacing	spread	spacing	spread
u_c_hihi.0	**321.9**	0.89	418.5	0.84	410.2	1.12	1095.7	**0.68**
u_c_hilo.0	**38.6**	0.91	50.3	0.94	61.8	1.00	204.4	**0.75**
u_c_lohi.0	87.0	0.77	**61.1**	0.91	64.5	0.55	221.1	**0.37**
u_c_lolo.0	7.4	0.87	**7.1**	0.96	11.0	0.96	23.0	**0.76**
u_i_hihi.0	354.4	0.83	321.6	**0.63**	**305.2**	0.72	494.8	0.82
u_i_hilo.0	41.9	0.88	**28.1**	0.99	47.1	0.89	99.9	**0.62**
u_i_lohi.0	57.9	0.87	58.1	0.89	**55.2**	0.91	109.2	**0.64**
u_i_lolo.0	8.3	**0.82**	**6.7**	0.99	7.4	1.30	23.6	0.98
u_s_hihi.0	357.3	**0.65**	385.8	0.87	**351.6**	0.67	769.2	0.50
u_s_hilo.0	**41.6**	1.10	46.4	0.88	45.0	0.91	74.7	**0.43**
u_s_lohi.0	**58.5**	0.74	75.4	1.19	61.5	0.69	176.5	**0.65**
u_s_lolo.0	10.6	1.04	**7.7**	0.77	8.4	1.01	16.8	**0.44**

Table 7.5: Diversity metrics of MOEAs for the multiobjective HCSP.

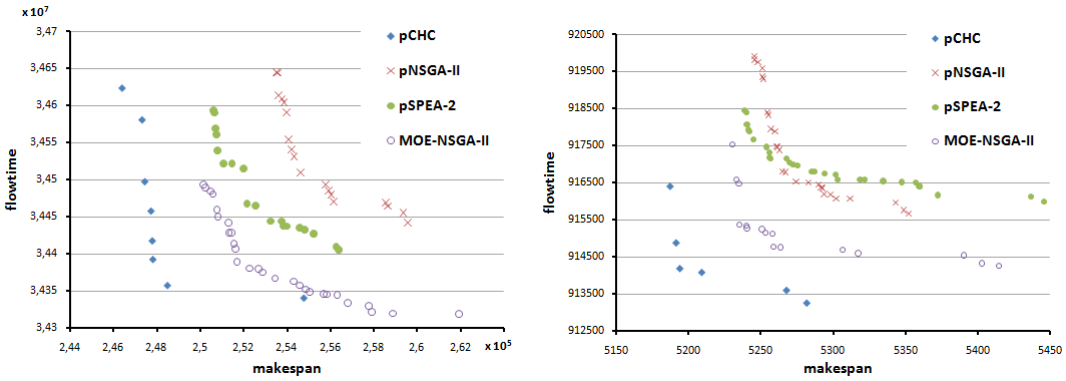
Regarding the efficiency metrics for multiobjective optimization, the results in Table 7.4 show that all the MOEAs –except pμ-CHC– were capable to find a reasonable number of non-dominated solutions for the twelve HCSP instances tackled, when using a population of 120 individuals. The pNSGA-II algorithm achieved a higher number of non-dominated solutions than the other MOEAs in more than half of the HCSP instances studied, while pSPEA-2 and MOE-NSGA-II usually obtained a slightly smaller number of non-dominated solutions. The aggregation approach in pμ-CHC frequently converged to a specific part of the Pareto front, obtaining a reduced number of non-dominated solutions (less than 10 non-dominated solutions for each problem instance, except for u_s_hihi.0). In contrast, pμ-CHC and MOE-NSGA-II obtained a significant largest number of Pareto-optimal solutions than the other two algorithms. The generational distance values confirm that pμ-CHC obtained the best solutions for the multiobjective HCSP, whose GD values have a significant difference with the other methods (in five out of twelve studied HCSP instances, the computed Pareto front is composed entirely by solutions found using pμ-CHC).

Concerning the diversity metrics for multiobjective optimization, the results in Table 7.5 show that pNSGA-II, pSPEA-2 and MOE-NSGA-II obtained similar values of the spacing metric. Instead, the pμ-CHC algorithm performed the worst, since it found few non-dominated solutions and was unable to correctly sample the region of the Pareto front associated to solutions with low flowtime metric values. These results are a possible consequence of the parameters (weights) used in the linear aggregation model for the fitness function in pμ-CHC. The weighted sum clearly defines an implicit trade-off between makespan and flowtime values and prevents pμ-CHC from computing schedules with low flowtime values. When considering the extreme points for each Pareto front in the calculation, the values of the spread metric show that despite it found few non-dominated solutions, the accuracy of the schedules computed by pμ-CHC is better than the results achieved with the other methods for nine out of twelve of the HCSP instances studied.

Figures 7.2, 7.3, and 7.4 present the Pareto fronts obtained for each evaluated MOEA for consistent, inconsistent and semiconsistent instances, respectively. Each Pareto front was built considering the non-dominated solutions found in the 30 independent executions performed for each algorithm. Due to the small values of the standard deviation found in the results and metrics for each MOEA, the Pareto front computed in a single execution of a specific MOEA does not differ significantly from the best results presented in the following figures.



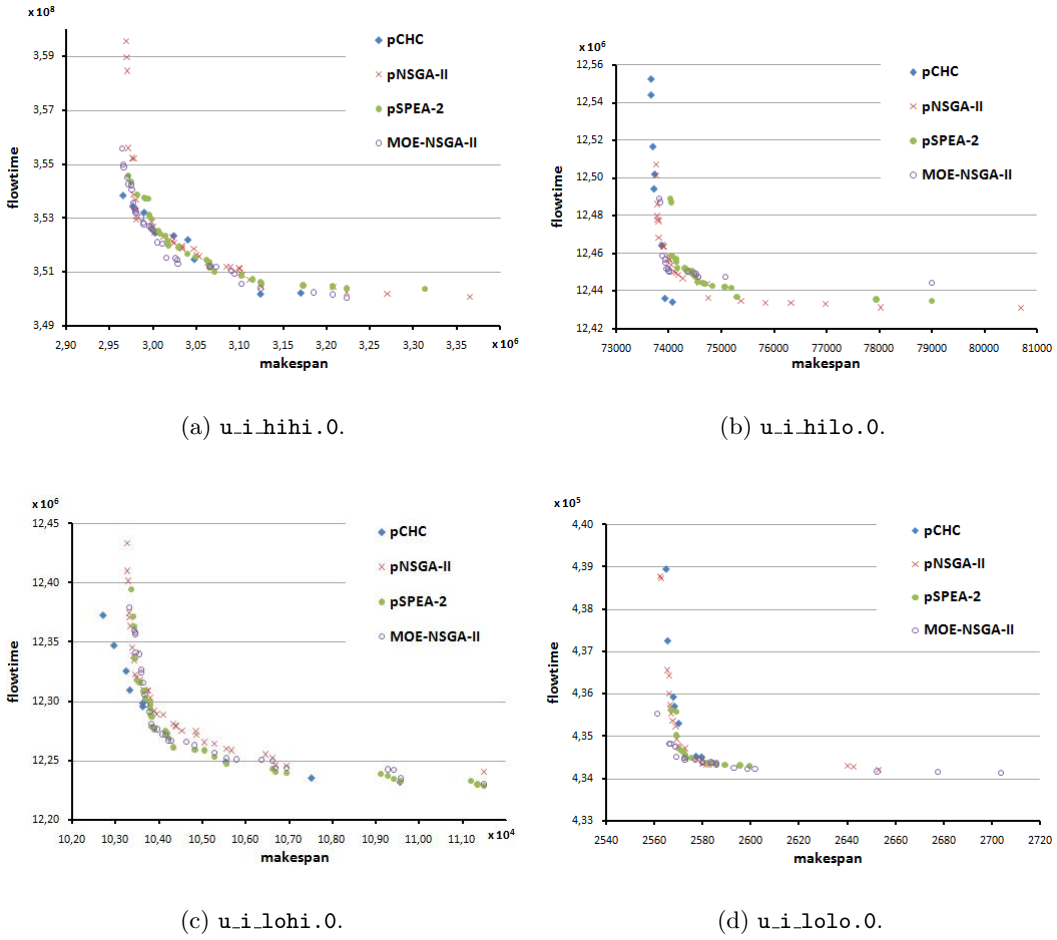
(a) `u_c_hihi.0`. (b) `u_c_hilo.0`.



(c) `u_c_lohi.0`. (d) `u_c_lolo.0`.

Figure 7.2: Pareto fronts for consistent HCSP instances.

The analysis of Figures 7.2, 7.3, and 7.4 shows a different behavior of the proposed algorithms for each type of problem instances solved. For the consistent HCSP instances, pμ-CHC obtained schedules with significantly better values of both makespan and flowtime metrics than the solutions achieved using the other MOEAs. These results confirm the usefulness of the PALS search for solving consistent scenarios, already detected for the single-objective HCSP. MOE-NSGA-II was able to compute quite accurate Pareto solutions, while pSPEA-2 and pNSGA-II performed the worst.



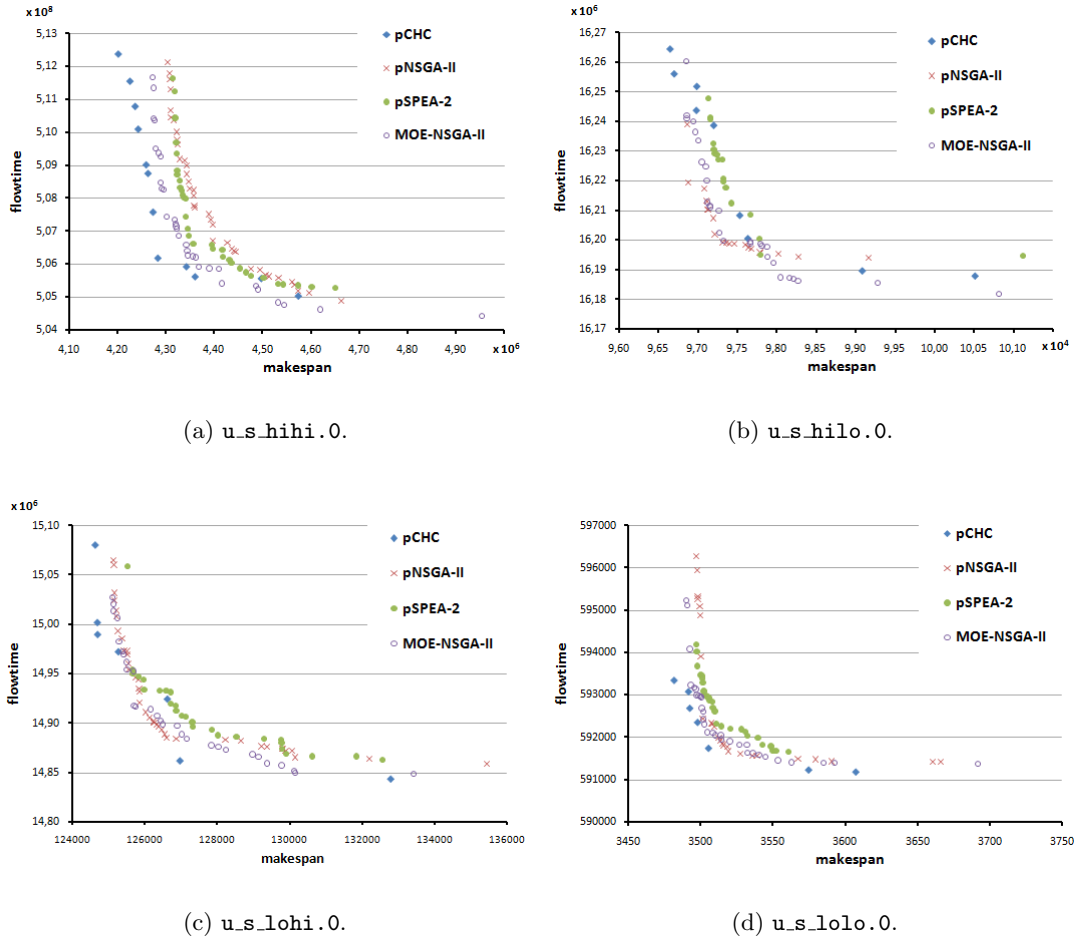
(a) `u_i_hihi.0`. (b) `u_i_hilo.0`.

(c) `u_i_lohi.0`. (d) `u_i_lolo.0`.

Figure 7.3: Pareto fronts for inconsistent HCSP instances.

All algorithms obtained similar results for inconsistent instances, accurately sampling the best compromise solutions, but pμ-CHC was unable to sample the zone of the Pareto front associated to schedules with low flowtime values. pNSGA-II and MOE-NSGA-II adequately sampled the Pareto front for all semiconsistent instances, while pμ-CHC was unable to compute precise solutions for `u_s_hihi.0`. In all cases the central region of each Pareto front –corresponding to interesting solutions with balanced compromise values between makespan and flowtime– was adequately sampled by all MOEAs.

Table 7.6 presents a comparison of the makespan and flowtime results obtained by MOE-NSGA-II and pμ-CHC –the best two methods among the studied MOEAs– against previous results reported for the two EAs previously applied to the makespan-flowtime optimization version of the HCSP: the cMA method by Xhafa et al. (2008a) and the hybrid combining MA and TS by Xhafa (2007). The previous work by Xhafa et al. (2008b), which applied a TS method for the makespan-flowtime multiobjective HCSP, has not been included in the comparison, since the flowtime results were not explicitly reported in the article by Xhafa et al. (2008b).



(a) `u_s_hihi.0`. (b) `u_s_hilo.0`.

(c) `u_s_lohi.0`. (d) `u_s_lolo.0`.

Figure 7.4: Pareto fronts for semiconsistent HCSP instances.

Instance	cMA		MA+TS		MOE-NSGA-II		pμ-CHC	
	makespan	flowtime	makespan	flowtime	makespan	flowtime	makespan	flowtime
`u_c_hihi.0`	7700929.8	1037049914.2	**7530020.2**	1046309158.0	7669870.0	1034100000.0	7595027.5	**1033251173.4**
`u_c_hilo.0`	155334.8	27487998.9	**153917.2**	27659089.9	154650.0	**27470600.0**	153997.5	27494220.4
`u_c_lohi.0`	251360.2	34454029.4	245288.9	34787262.8	250148.0	**34319000.0**	**246392.7**	34340362.5
`u_c_lolo.0`	5218.2	913976.2	**5173.7**	920222.3	5230.9	914269.0	5187.4	**913252.5**
`u_i_hihi.0`	3186664.7	361613627.3	3058474.9	368332234.0	**2964590.0**	**350056000.0**	2966065.8	350190000.5
`u_i_hilo.0`	75856.6	12572126.6	75108.5	12757607.2	73825.1	12433400.0	**73674.4**	12433973.2
`u_i_lohi.0`	110620.8	12707611.5	105808.6	12912987.9	103336.0	12230400.0	**102732.6**	12231807.1
`u_i_lolo.0`	2624.2	439073.7	2596.6	444764.9	**2561.5**	**434148.0**	2564.7	434524.3
`u_s_hihi.0`	4424540.9	513769399.1	4321015.4	532319945.0	4273710.0	**504447000.0**	**4201963.5**	505027968.6
`u_s_hilo.0`	98283.7	16300484.9	97177.3	16616505.4	96855.9	**16181900.0**	**96649.5**	16187925.1
`u_s_lohi.0`	130014.5	15179363.5	127633.0	15743720.0	125121.0	14848700.0	**124650.2**	**14843952.2**
`u_s_lolo.0`	3522.1	594666.0	3484.1	604519.1	3490.2	591381.0	**3482.1**	**591169.6**

Table 7.6: Comparison of MOE-NSGA-II and pμ-CHC against previous evolutionary techniques for the multiobjective HCSP.

The comparative analysis of the results presented in Table 7.6 shows that MOE-NSGA-II and pμ-CHC were able to compute accurate compromise solutions for the multiobjective HCSP, outperforming the previous results obtained with evolutionary techniques for the problem (the best results for each metric are marked with bold font in Table 7.6). The MOE-NSGA-II and pμ-CHC algorithms obtained better solutions than the previous cMA and MA+TS, except for the makespan metric in consistent instances. As well as for the single-objective HCSP, consistent problems appear to be the most difficult class of instances to solve with the multiobjective approach proposed in this work.

The previously reported results suggest that using explicit multiobjective EAs is a promising idea for solving the multiobjective version of the HCSP aimed at minimizing makespan and flowtime. The reported results could be further improved by using flowtime-oriented evolutionary operators, which were not explicitly included in the implemented MOEAs.

7.3.7 Concluding remarks

The analysis of EAs applied to the multiobjective version of the HCSP showed that the MOE-NSGA-II algorithm and the pμ-CHC algorithm using the aggregation approach are promising methods to simultaneously optimize the makespan and flowtime metrics. The explicit multiobjective approach applied in MOE-NSGA-II was capable to obtain accurate solutions for the problem instances by Braun et al. (2001), achieving schedules with good compromise values between the makespan and flowtime objectives and a correct sampling of the Pareto front. MOE-NSGA-II and pμ-CHC obtained better solutions than previous single-objective EAs applied to the problem, improving the makespan and flowtime metrics for all problem instances by Braun et al. (2001), except for three out of four consistent instances. These results suggest that the proposed MOEAs are useful schedulers for HC systems when considering the makespan and flowtime metrics to optimize. Further work is needed in order to confirm the suitability of the proposed MOEAs for solving large HCSP instances (several non-formalized experiments have been successfully performed using the new set of large-sized HCSP instances, but the results are still non-conclusive). Using flowtime-oriented exploration operators and including a Pareto-based fitness assignment schema in pμ-CHC are clear lines of future work in order to improve the flowtime results and the population diversity of the new pμ-CHC method applied to the problem.

7.4 Summary

This chapter presented the advances on applying the new pμ-CHC algorithm to solve two HCSP variants: a dynamic HCSP version applying the rescheduling strategy and a multiobjective HCSP version that simultaneously optimizes the makespan and flowtime metrics.

The initial studies showed that pμ-CHC is a promising method for solving both HCSP variants tackled. The pμ-CHC algorithm using the rescheduling strategy significantly improves over a traditional pμ-CHC, the traditional Min-Min, and a Min-Min using the rescheduling strategy, specially when scheduling a large number of tasks. The makespan improvements were up to **8%** over the traditional pμ-CHC and above **19%** over the Min-Min schedulers.

The previously commented results suggest that pμ-CHC is able to efficiently handle dynamic HC scenarios by using the rescheduling strategy. On the other hand, the analysis of EAs applied to the multiobjective version of the HCSP showed that pμ-CHC and the MOE-NSGA-II methods were able to obtain accurate schedules regarding the makespan and flowtime metrics. Both methods outperformed the best solutions previously obtained with single-objective EAs applied to the problem.

Further work and experimental analysis are needed in order to clearly determine the benefits of the rescheduling strategy and comparing the rescheduling pμ-CHC with other dynamic and on-line methods for scheduling. Further work is also needed to study the efficacy and efficiency of the proposed MOEAs for solving large HCSP instances, possibly using flowtime-oriented exploration operators and including a Pareto-based fitness assignment schema in pμ-CHC.

Chapter 8

Conclusions and future work

This chapter presents the conclusions of the work on applying sequential and parallel EAs to the HCSP. It also formulates the main lines for future work that arose during the preparation of this thesis.

8.1 Conclusions

This work presented the application of sequential and parallel EAs for solving the scheduling problem in HC and grid computing environments.

The last decade saw a massive popularization of distributed computing for cooperatively solving complex problem. So, the HCSP became a capital problem when executing tasks in distributed heterogeneous computing and grid platforms. In its classical formulation, the HCSP assumes the minimization of the makespan metric, which evaluates the time spent from the moment when the first task starts its execution to the moment when the last task is completed. The static version of the problem was tackled, since it adequately models the planning in distributed clusters and HC multiprocessors, while it also provides a first step for solving more complex scheduling problems arising in distributed and dynamic environments. The independent task model was used because it captures the reality of most distributed and grid computing environments where different users submit their (obviously independent) tasks to execute in a HC cluster or grid computing service.

The proposed EAs were designed to find accurate HCSP solutions in an efficient way, using a bounded time stopping criterion that allows a quick planning and eventually the rescheduling of incoming tasks. Fast parallel versions of GA and CHC were designed aiming at exploiting both the intrinsic parallel nature of EAs and the resource availability in distributed computing environments and modern multicore computers.

The EAs were implemented using the MALLBA library and they were executed in a high performance cluster for solving a large set of HCSP instances. In a first stage, the accuracy of sequential and parallel EAs to solve a low-sized standard test suite of HCSP instances was studied, before scaling up to solve larger scenarios. In a second stage, a new set of large-sized HCSP instances was introduced, in order to model realistic distributed HC and grid scenarios, specially relevant to analyze the scalability behavior of the proposed EAs. The test suite comprises several problem instances that were designed following a methodology based on the well-known ETC model for execution time estimation by Ali et al. (2000).

The first experimental analysis allowed drawing some conclusions about the applicability of EAs to solve the HCSP. The traditional GA is not well-suited to solve the HCSP efficiently, mainly because its slow fitness evolution conspires against finding accurate results in short times. CHC is a more adequate strategy for solving the HCSP when compared with a traditional GA model. The parallel EAs significantly improved the results of the sequential algorithms taking advantage of their multiple search mechanism and the diversity provided by the subpopulation model. The parallel algorithms were able to exploit working with small populations to improve the search efficacy and efficiency by using the available computing power in a parallel-distributed environment. The parallel CHC method showed good trade-off values when considering the solution quality and the execution time required to compute it. Despite starting from suboptimal schedules, it was able to find well-suited schedules in a reasonable low number of generations, and was able to improve the makespan values faster than the other EAs. The pCHC method was capable of improving upon the previous best-known makespan results for six instances out of twelve in the problem set by Braun et al. (2001), a remarkable efficiency result when considering the exhaustive work done by previous researchers to solve this small-sized HCSP test suite.

The scalability results showed that pCHC has a more focused exploration pattern when using smaller populations. However, the loss of diversity caused the makespan results to deteriorate when splitting the population in demes with less than 15 individuals, suggesting that further improvements were needed in order to achieve a fully scalable scheduler. These results lead to the design of the new pμ-CHC algorithm.

The new algorithmic proposal in pμ-CHC is inspired by concepts from multiobjective EAs, and it is aimed at exploiting both the intrinsic parallel nature of EAs and the resource availability in grid environments. The pμ-CHC algorithm uses a small population of eight individuals and follows an accelerated evolution model, using a powerful randomized local search method based in PALS to enhance the search. The new method was implemented using the MALLBA library, employing a two-level parallel model that uses shared memory and message passing communication paradigms. The numerical analysis allowed to conclude that pμ-CHC is an efficient scheduler for HC and grid environments, which is able to obtain accurate schedules in reduced execution times. pμ-CHC is the new state-of-the-art algorithm for the benchmark set of twelve HCSP instances by Braun et al. (2001), since it was able to improve over the previous best-known solutions achieved with diverse metaheuristic techniques.

Experiments performed over the new large-sized HCSP instances provided a first step toward a grid-level scalability analysis of pCHC and pμ-CHC for solving the HCSP on realistic grid scenarios up to 256 processors and 8192 tasks. When solving those high dimension HCSP instances, pμ-CHC was also capable of computing accurate results with respect to those obtained using deterministic heuristics and the pCHC algorithm. The makespan improvement factors attained by pμ-CHC were **15%** with respect to the Min-Min heuristic, and **20%** (more than **25%** for the largest instances) with respect to the Sufferage heuristic. Lower improvements were obtained when comparing against the pCHC method, but the improvements increased as the instances grew, achieving a **3.5%** overall improvement factor for HCSP instances with dimension 8192×256 (over **4%** for consistent and semi-consistent instances, overcoming the problem of pCHC to deal with structured scenarios). The scalability and parallel performance analysis showed that pμ-CHC is able to improve the makespan results when using more computing resources.

The experimental analysis showed that the pμ-CHC algorithm successfully solved HCSP instances with up to thousands of tasks and several hundred of machines. From the previous results, the main conclusion of this work is the assertion that pμ-CHC is a powerful tool for scheduling in large distributed HC and grid environments when dealing with tasks having large execution times. In these scenarios, it is worth to invest the time required for computing the schedule (i.e. less than two minutes) in order to achieve significant improvements (over **15%**) in the makespan values over traditional deterministic heuristics.

Summarizing, the main contributions of the research reported in this thesis are:

- A conceptualization of the scheduling problem in heterogeneous computing environments has been presented.
- Parallel evolutionary algorithms have been described, and the main features that make them useful for finding accurate solutions for the HCSP have been identified.
- A new parallel micro CHC algorithm, aimed at computing accurate schedules for HCSP scenarios of increasing complexity, has been introduced.
- A comprehensive survey of related works that have proposed to apply EAs for solving the HCSP was provided.
- Sequential and parallel EAs have been applied to the HCSP. The new pμ-CHC was identified as the state-of-the-art method for solving the HCSP, outperforming the previously best-known methods for standard HCSP instances and also demonstrating a good scalability behavior when solving large HCSP instances. The main experimental results are summarized in Table 8.1.
- Two variants of the HCSP (scheduling in dynamic scenarios and multiobjective HCSP) have been initially studied. The analysis showed that parallel EAs are able to compute accurate solutions for both problems variants.

metric	**scenarios**	**main results (pμ-CHC)**
makespan	512×16 (Braun et al.)	new best solutions for all instances
	new large instances	**15-20%** of improvement over deterministic heuristics **3.5%** of improvement over pCHC
GAP(LB)	512×16 (Braun et al.)	GAP(LB) below **2%** more than **94.5%** of the ideal improvement
	new large instances	GAP(LB) below **6.20%** more than **71%** of the ideal improvement
execution time	512×16 (Braun et al.)	5% of improvement in less than **10 s.** 10% of improvement in **15 s.**
	new large instances	5% of improvement in **45 s.** 10% of improvement in **85 s.**
scalability	512×16 (Braun et al.)	**5%** of normalized makespan reduction
	new large instances	up to **10%** of normalized makespan reduction
parallel performance	all scenarios	best results when using additional demes

Table 8.1: Summary of the main experimental results.

8.2 Future work

The main lines of future work identified during the research are already in progress, and some preliminary results were already presented in Chapter 7.

An important issue from a practical point of view involves studying the applicability of pμ-CHC and other parallel EAs for solving dynamic versions of HCSP by using a rescheduling strategy. The pμ-CHC method is a powerful tool for quickly scheduling a bunch of tasks; thus an iterated version of the algorithm could be used to perform rescheduling on dynamic scenarios every time that specific events occur (such as new tasks arriving, adding or deleting computing resources, etc.). Specifically, a simple dynamic version of the HCSP was tackled, which proposes facing dynamic environments using the rescheduling strategy. The preliminary analysis performed on a set of three dynamic HCSP instances with up to 50000 tasks and 100 machines showed that pμ-CHC using the rescheduling strategy was able to deal with dynamic scenarios. The reported results show that significant makespan improvements can be obtained when using the rescheduling pμ-CHC, specially in those HCSP scenarios involving the execution of a large number of tasks. Further work and experimental analysis are needed in order to clearly determine the benefits of the rescheduling strategy and compare the rescheduling pμ-CHC with other dynamic and on-line methods for scheduling in HC environments.

In addition, a multiobjective version of the HCSP that proposes minimizing the makespan and flowtime objectives was solved with the pμ-CHC method and several MOEAs. Despite using a single-objective approach that combines these two objectives using a linear aggregation function, pμ-CHC was able to obtain the best solutions when considering the compromise values of makespan and flowtime. However, the linear aggregation approach in pμ-CHC frequently converged to the low-makespan section of the Pareto front, seldom sampling the low-flowtime section. Including a Pareto-based fitness assignment schema in pμ-CHC is a certain line of future work in order to improve the flowtime results and the population diversity of the new pμ-CHC method. A parallel implementation of the NSGA-II algorithm obtained the best sampling patterns for the Pareto front of the problem, demonstrating that MOEAs are suitable to solve the HCSP version faced. Further work needs to be done in order to improve the results, possibly by using flowtime-oriented evolutionary operators –which were not explicitly used in the implemented MOEAs– and also to confirm the suitability of the proposed MOEAs for solving large HCSP instances.

These lines of work are currently being investigated.

On the other hand, some ideas for future work have not been presented in Chapter 7. A third line of future work should focus on improving the efficacy and efficiency of the proposed methods, exploring the ability of pμ-CHC and the other parallel EAs to face scenarios that model still larger HC environments than the ones studied in this work. In order to face those larger scenarios, new compact problem encodings and evolutionary operators that perform large schedule modifications need to be devised to overcome the slow evolution pattern for large instances. A new scalable pμ-CHC algorithm, capable of improving the search speed by using the computational power of large clusters of machines, could be applied to efficiently solve HCSP instances with more than a thousand machines.

From a different point of view, an additional line of research involves studying the fitness landscape for the HCSP. Following the recent theoretical works by Whitley et al. (2008) and Whitley and Sutton (2009), it is possible to provide a theoretical character-

ization of any optimization problem aimed at modeling the behavior of a local search method to solve it. If the HCSP can be classified into the *elementary landscape* class of problems, new ideas can be devised in order to design powerful local search operators for the problem. Furthermore, if the recent results demonstrated for local search operators for the Traveling Salesman Problem could be extended to the HCSP, the solution space could be explored without performing an explicit search. This line of research would lead to a novel theoretical analysis of the HCSP and related scheduling problems, and also provide a promising advance for tackling large-sized problem instances involving thousands of tasks and machines.

Appendix A

HCSP instances generator and test suites

This appendix presents the code of the generator program used to create the new large-sized HCSP instances introduced in this thesis. After that, the parametrization used to create the large-sized HCSP instances are presented, in order to allow the replication of the experiments performed in this project.

A.1 HCSP instances generator

The HCSP instances generator program is implemented in the C language using the standard C libraries `stdlib.h` and `math.h`, without requiring any additional software. The generator implements the range based method from Ali et al. (2000), regarding the relevant scenario parameters: dimension (number of tasks and machines), task and machine heterogeneity, consistency, and two different parametrization models of ETC.

The following code implements the HCSP instances generator program.

```
// Generator program for HC and grid scheduling.
// Implements the range-based method by Ali et al. (2000)
//
// Output file format: similar to the one used by Braun et al. (2001), column vector used
//   to represent the expected time to compute matrix ETC(t_i,m_j):
//   floating point numbers, dimension (TXM)x1, ordered by task identifier
//   (t1,m1)\\(t1,m2)\\...\\(t1,mM)\\(t2,m1)\\(t2,m2)...\\(tT,m1)\\...\\(tT,mM).
// The first line in the input file specifies the number of tasks and machines in the
//   generated scenario.
// Input parameters:
//   Number of tasks, number of machines, task heterogeneity level (0-Low, 1-High),
//   machine heterogeneity level (0-Low, 1-High), consistency type (0-consistent,
//   1-semiconsistent, 2-inconsistent).
// Optional parameters:
//   [heterogeneity model: 0-Ali et al., 1-Braun et al.] [variable type: 0-real, 1-integer].
//   Heterogeneity ranks (tasks, machines): Ali et al. (10-100000,10-100), Braun et al.
//   (100-3000,10-1000).
//   Heterogeneity model by Braun et al. (100-3000,10-1000) assumed by default.
//
// Bibliographic references
// Ali, S., Siegel, H. J., Maheswaran, M., Ali, S., and Hensgen, D. (2000). Task Execution
//   Time Modeling for Heterogeneous Computing Systems. In Proceedings of the 9th
//   Heterogeneous Computing Workshop. IEEE Computer Society, Washington, DC, pp. 185.
```

```
// Braun, T. D., Siegel, H. J., Beck, N., Blni, L. L., Maheswaran, M., Reuther, A. I.,
//   Robertson, J. P., Theys, M. D., Yao, B., Hensgen, D., and Freund, R. F. (2001). A
//   comparison of eleven static heuristics for mapping a class of independent tasks onto
//   heterogeneous distributed computing systems. Journal of Parallel and Distributed
//   Computing 61, 6, pp. 810-837.

#include <math.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]){

if (argc < 6){
    printf("Sintaxis: %s <num_tasks> <num_machines> <task_heterogeneity> <machine_heterogeneity>
        <consistency> [model] [type] [seed]\n", argv[0]);
    printf("Task heterogeneity levels: (0-Low, 1-High), machine heterogeneity levels:
        (0-Low, 1-High).\n");
    printf("Consistency type: (0-Consistent, 1-Semiconsistent, 2-Inconsistent).\n");
    printf("Optional: heterogeneity model: (0-Ali et al., 1-Braun et al.).\n");
    printf("\tRanks (tasks, machines) 0(Ali):(10-100000,10-100), 1(Braun):(100-3000,10-1000).\n");
    printf("\t(ranks by Braun et al. (100-3000,10-1000) assumed by default).\n");
    printf("Optional: type of task execution times: (0-real, 1-integer).\n");
    printf("Optional: seed for the pseudorandom number generator.\n");

    exit(1);
}

char * ht_st; // "hi" or "lo"
char * hm_st; // "hi" or "lo"
char consist;  // "c": consistent, "u": inconsistent, "s" semiconsistent.
char mod;      // "A": Ali et al. (2000), "B": Braun et al. (2001).
char dist;     // "u": uniform.

int RT, RM, model
char type;

ht_st = (char *) malloc (sizeof(char)*2);
hm_st = (char *) malloc (sizeof(char)*2);

if (argc > 6){
    model = atoi(argv[6]);
    if (model == 0){
        mod = 'A';
    } else {
        if (model > 1){
            printf("Heterogeneity model by Braun et al. (2001) assumed by default\n");
        }
        mod='B';
    }
} else {
    model = 1;
    mod='B';
    printf("Heterogeneity model by Braun et al. (2001) assumed by default\n");
}

if (argc > 7){
    type = atoi(argv[7]);
```

```
    if (type == 1){
        printf("Type of task execution times: integer.\n");
    } else {
        type = 0;
        printf("Type of task execution times: real.\n");
    }
} else {
    type = 0;
    printf("Type of task execution times: real (assumed by default).\n");
}

int seed;

if (argc > 8){
    printf("argv_8: %s\n",argv[8]);
    seed = atoi(argv[8]);
    printf("seed: %d\n",seed);
} else {
    seed=time(NULL);
}

// Initialization of the pseudorandom number generator.
srand48(seed);

int NT = atoi(argv[1]);
int NM = atoi(argv[2]);

dist = 'u'; // Uniform distribution.

int HT = atoi(argv[3]);
if (HT == 0){
    strcpy(ht_st,"lo");
    if (model == 0){
        RT = 10;
    } else {
        RT = 100;
    }
} else {
    strcpy(ht_st,"hi");
    if (HT > 1){
        printf("Task heterogeneity level: 1-High (by default).\n");
        HT = 1;
    }
    if (model == 0){
        RT = 100000;
    } else {
        RT = 3000;
    }
}

int HM = atoi(argv[4]);
if (HM == 0){
    strcpy(hm_st,"lo");
    RM = 10;
} else {
    strcpy(hm_st,"hi");
    if (HM > 1){
        printf("Machine heterogeneity level: 1-High (by default).\n");
        HM = 1;
```

```
    }
    if (model == 0){
        RM = 100;
    } else {
        RM = 1000;
    }
}

int cons = atoi(argv[5]);
if (cons == 0){
    consist='c';
} else if (cons == 1){
    consist='s';
} else {
    if (cons > 2){
        printf("Consistency type: 2-Inconsistent (by default).\n");
    }
    cons = 2;
    consist='i';
}

int i,j;
float **ETC = (float **) malloc(sizeof(float *)*NT);

if (ETC == NULL){
    printf("Error in memory allocation for ETC, dimension %dx%d\n",NT,NM);
    exit(2);
}

for (i=0;i<NT;i++){
    ETC[i] = (float *) malloc(sizeof(float)*NM);
    if (ETC[i] == NULL){
        printf("Error in memory allocation for row %d of ETC.\n",i);
        exit(2);
    }
}

char file_out[15];
char file_log[15];

sprintf(file_out,"%c.%c_%c_%s%s",mod,dist,consist,ht_st,hm_st);
printf("file out: [%s]\n",file_out);

sprintf(file_log,"%s.log",file_out);

FILE *fp;
FILE *fl;

if((fl=fopen(file_log, "w"))==NULL){
    printf("Cannot write in file %s\n",file_log);
    exit(1);
}

fprintf(fl,"File: [%s]\n",file_out);
fprintf(fl,"Test scenario with %d tasks and %d machines\n",NT,NM);
fprintf(fl,"Model %c, distribution %c, consistency %c\n",mod,dist,consist);
fprintf(fl,"RT:%d, RM:%d\n",RT,RM);
fprintf(fl,"Task heterogeneity: %s, Machine heterogeneity: %s\n",ht_st,hm_st);
```

```
if((fp=fopen(file_out, "w"))==NULL){
    printf("Cannot write in file %s\n",file_out);
    exit(1);
}

if (type == 1){
    // ETC: integer values.
    fprintf(fl,"Type of task execution times: integer.\n");
} else {
    fprintf(fl,"Type of task execution times: real.\n");
}

fprintf(fl,"seed: [%d]\n",seed);

float p;
float *fila = (float *)malloc(sizeof(float)*NM);

for (i=0;i<NT;i++){
    p = RT*drand48();

    for (j=0;j<NM;j++){
        fila[j] = p*RM*drand48();
    }
    // Consistent instance: reordering of rows is required.
    if (cons == 0){
        // Sort row
        quickSort(fila,0,NM-1);
    } else {
        if (cons == 1){
        // Semiconsistent instance: reordering of odd rows is required.
            if ( (i % 2) == 0 ){
                quickSort(fila,0,NM-1);
            }
        }
    }

    for (j=0;j<NM;j++){
        ETC[i][j] = fila[j];
    }
}

// The first row in the file specifies the number of tasks and machines in the scenario.
fprintf(fp,"%d %d\n",NT,NM);

for (i=0;i<NT;i++){
    for (j=0;j<NM;j++){
        if (type == 1){
            // ETC: real values
            fprintf(fp,"%.0f\n",floor(ETC[i][j]));
        } else {
            // ETC: integer values
            fprintf(fp,"%.2f\n",ETC[i][j]);
        }
    }
}

close(fp);
close(fl);
}
```

A.2 Test suites

The set of low-sized de-facto standard HCSP by Braun et al. (2001) was provided by F. Xhafa. The problem set is publicly available to download at the HCSP website (download section) `http://www.fing.edu.uy/inco/grupos/cecal/hpc/HCSP/download`.

The new large-sized HCSP instances introduced in this thesis were generated using the previously presented generator program, executed with the seeds for the pseudorandom number generator presented in the following tables.

Files containing the seeds used to generate the new large-sized HCSP instances are also publicly available to download at the HCSP website (download section).

dimension:512×16 (heterogeneity parameters by Ali et al. (2000))
A.u_c_hihi, seed: [1245940097]
A.u_c_hilo, seed: [1245940161]
A.u_c_lohi, seed: [1245940157]
A.u_c_lolo, seed: [1245940165]
A.u_i_hihi, seed: [1245940197]
A.u_i_hilo, seed: [1245940194]
A.u_i_lohi, seed: [1245940189]
A.u_i_lolo, seed: [1245940192]
A.u_s_hihi, seed: [1245940177]
A.u_s_hilo, seed: [1245940174]
A.u_s_lohi, seed: [1245940180]
A.u_s_lolo, seed: [1245940170]

dimension: 1024×32	dimension: 2048×64
A.u_c_hihi, seed: [1214388943]	A.u_c_hihi, seed: [1214389178]
A.u_c_hilo, seed: [1214388934]	A.u_c_hilo, seed: [1214389175]
A.u_c_lohi, seed: [1214388952]	A.u_c_lohi, seed: [1214389170]
A.u_c_lolo, seed: [1214389061]	A.u_c_lolo, seed: [1214389159]
A.u_i_hihi, seed: [1214389090]	A.u_i_hihi, seed: [1214389262]
A.u_i_hilo, seed: [1214389023]	A.u_i_hilo, seed: [1214389259]
A.u_i_lohi, seed: [1214389052]	A.u_i_lohi, seed: [1214389253]
A.u_i_lolo, seed: [1214389075]	A.u_i_lolo, seed: [1214389256]
A.u_s_hihi, seed: [1214389037]	A.u_s_hihi, seed: [1214389217]
A.u_s_hilo, seed: [1214389032]	A.u_s_hilo, seed: [1214389221]
A.u_s_lohi, seed: [1214389040]	A.u_s_lohi, seed: [1214389227]
A.u_s_lolo, seed: [1214388834]	A.u_s_lolo, seed: [1214389224]
B.u_c_hihi, seed: [1214388890]	B.u_c_hihi, seed: [1214389181]
B.u_c_hilo, seed: [1214388893]	B.u_c_hilo, seed: [1214389187]
B.u_c_lohi, seed: [1214388886]	B.u_c_lohi, seed: [1214389194]
B.u_c_lolo, seed: [1214389064]	B.u_c_lolo, seed: [1214389191]
B.u_i_hihi, seed: [1214388851]	B.u_i_hihi, seed: [1214389267]
B.u_i_hilo, seed: [1214388857]	B.u_i_hilo, seed: [1214389271]
B.u_i_lohi, seed: [1214388846]	B.u_i_lohi, seed: [1214389278]
B.u_i_lolo, seed: [1214388841]	B.u_i_lolo, seed: [1214389281]
B.u_s_hihi, seed: [1214388865]	B.u_s_hihi, seed: [1214389214]
B.u_s_hilo, seed: [1214388861]	B.u_s_hilo, seed: [1214389211]
B.u_s_lohi, seed: [1214388871]	B.u_s_lohi, seed: [1214389207]
B.u_s_lolo, seed: [1214388875]	B.u_s_lolo, seed: [1214389204]

dimension: 4096×128	dimension: 8192×256
A.u_c_hihi, seed: [1240440634]	A.u_c_hihi, seed: [1241443018]
A.u_c_hilo, seed: [1240440631]	A.u_c_hilo, seed: [1241443031]
A.u_c_lohi, seed: [1240440639]	A.u_c_lohi, seed: [1241443049]
A.u_c_lolo, seed: [1240440550]	A.u_c_lolo, seed: [1241443036]
A.u_i_hihi, seed: [1240440654]	A.u_i_hihi, seed: [1241443011]
A.u_i_hilo, seed: [1240440649]	A.u_i_hilo, seed: [1241442991]
A.u_i_lohi, seed: [1240440643]	A.u_i_lohi, seed: [1241442999]
A.u_i_lolo, seed: [1240440646]	A.u_i_lolo, seed: [1241442981]
A.u_s_hihi, seed: [1240440656]	A.u_s_hihi, seed: [1241034415]
A.u_s_hilo, seed: [1240440659]	A.u_s_hilo, seed: [1241442958]
A.u_s_lohi, seed: [1240440665]	A.u_s_lohi, seed: [1241442966]
A.u_s_lolo, seed: [1240440662]	A.u_s_lolo, seed: [1241442977]
B.u_c_hihi, seed: [1240440738]	B.u_c_hihi, seed: [1241443021]
B.u_c_hilo, seed: [1240440735]	B.u_c_hilo, seed: [1241443026]
B.u_c_lohi, seed: [1240440730]	B.u_c_lohi, seed: [1241443043]
B.u_c_lolo, seed: [1240440732]	B.u_c_lolo, seed: [1241443040]
B.u_i_hihi, seed: [1240440701]	B.u_i_hihi, seed: [1241443006]
B.u_i_hilo, seed: [1240440707]	B.u_i_hilo, seed: [1241442988]
B.u_i_lohi, seed: [1240440724]	B.u_i_lohi, seed: [1241443001]
B.u_i_lolo, seed: [1240440710]	B.u_i_lolo, seed: [1241442984]
B.u_s_hihi, seed: [1240440687]	B.u_s_hihi, seed: [1241445656]
B.u_s_hilo, seed: [1240440679]	B.u_s_hilo, seed: [1241442950]
B.u_s_lohi, seed: [1240440669]	B.u_s_lohi, seed: [1241442969]
B.u_s_lolo, seed: [1240440676]	B.u_s_lolo, seed: [1241442974]

Appendix B

Best solutions for the HCSP test suite from Braun et al. (2001)

This appendix presents the best solutions (schedules) found using pCHC and pμ-CHC for the HCSP instances in the test suite by Braun et al. (2001).

For the sake of simplicity, the solutions are presented using the task-oriented representation (an array of machine identifiers, where the presence of m_j in the position t_i means that the task t_i is scheduled to execute on machine m_j, see Section 5.2).

The best solutions are also publicly available to download at the HCSP website (results section) `http://www.fing.edu.uy/inco/grupos/cecal/hpc/HCSP/HCSP_res`. The best solutions obtained for the new large-sized HCSP instances specifically designed in this work are also presented in the HCSP website. They have been omitted here due to the large space required to present it.

pCHC

Instance: `u_c_lohi.0`, makespan: **241524.0** Solution: [0 0 1 0 12 11 2 0 0 5 2 9 1 1 15 2 0 0 0 0 3 9 0 1 4 0 1 0 3 14 0 0 9 6 4 0 0 0 0 6 0 0 0 9 14 12 1 0 2 1 0 0 7 1 1 0 1 4 3 0 2 1 8 13 1 1 1 0 0 1 14 2 7 5 0 6 0 0 1 0 0 4 0 0 6 0 1 0 2 0 0 0 0 0 2 3 14 1 2 15 0 5 1 11 0 1 1 1 14 6 1 1 1 2 0 0 0 4 1 0 4 0 0 0 0 6 7 1 5 3 4 0 0 0 1 0 6 0 8 13 1 0 0 2 7 2 0 5 0 4 0 2 2 1 8 0 0 0 1 1 4 5 8 12 0 0 0 5 3 0 0 0 5 6 2 1 0 0 0 9 1 2 0 2 3 6 6 0 3 1 0 0 2 0 4 0 5 0 1 3 0 12 2 3 0 3 0 1 0 0 2 0 2 0 0 0 7 0 0 2 0 11 0 6 0 3 0 0 5 2 0 0 0 0 1 0 10 13 1 8 8 0 4 0 0 0 0 6 0 3 2 2 3 0 5 3 0 3 13 9 13 9 0 10 0 0 4 0 0 5 1 0 1 2 0 11 0 0 3 0 0 15 3 4 9 0 12 0 5 4 7 8 0 0 0 0 2 3 7 0 7 0 0 7 14 1 2 11 0 2 0 0 1 14 1 0 3 0 9 14 1 5 0 0 2 0 0 0 4 7 1 1 2 0 2 1 0 0 0 2 7 0 5 0 0 0 0 10 2 3 0 15 1 0 1 0 0 3 1 0 0 13 0 0 3 0 0 0 9 1 0 0 3 0 0 0 0 0 0 2 0 13 0 0 1 8 0 0 1 11 1 1 2 2 2 0 0 1 0 10 4 1 13 10 0 0 4 0 3 6 2 0 2 4 0 0 0 7 4 0 0 4 0 8 6 0 0 4 3 7 0 1 2 0 0 15 0 0 10 2 1 0 0 0 0 1 15 10 1 0 9 4 0 8 0 0 7 0 7 6 0 0 8 2 0 1 1 0 0 3 1 0 0 0 0 0 8 3 2 5 1 1 0 0 2 8 0 1 3 0 0 9 0 0 5 0 9 0 0 10 8 2 3 0 0 4 0 0 0 0 2 0]

Instance: `u_i_hilo.0`, makespan: **73639.8** Solution: [4 2 0 13 10 8 14 7 13 13 0 4 1 11 6 8 15 5 13 0 6 1 11 14 3 4 12 11 7 15 1 5 12 12 10 10 4 12 8 8 12 11 3 11 5 10 5 8 6 4 3 8 14 11 6 0 11 2 9 15 7 8 9 11 11 8 2 3 9 12 4 4 15 0 3 9 14 8 13 2 15 15 11 2 8 10 12 10 3 11 0 8 2 7 9 4 2 4 14 15 13 8 9 4 6 15 11 14 1 13 1 12 1 0 6 0 9 7 13 7 12 8 1 12 8 10 7 12 4 2 0 12 3 0 8 7 12 4 6 4 10 2 3 1 7 3 0 14 11 1 12 11 2 15 3 2 10 8 1 0 0 14 10 1 1 2 0 3 5 13 6 1 11 0 1 10 1 0 6 12 6 0 12 2 6 11 0 2 9 6 4 7 0 3 1 0 3 1 11 9 15 8 4 13 14 6 0 0 6 12 6 12 11 9 3 10 12 9 15 6 7 10 10 13 14 9 0 5 3 15 0 11 12 3 0 11 0 11 6 3 6 5 9 10 2 7 10 4 13 9 15 6 12 0 5 13 12 13 10 14 8 9 0 5 0 1 0 10 10 13 13 7 13 12 6 7 12 1 2 15 10 12 6 5 13 10 10 5 15 13 15 3 12 1 14 11 1 0 13 14 11 3 12 3 6 9 4 6 15 11 1 6 13 3 11 12 6 14 9 5 8 3 10 8 5 0 10 8 1 1 1 9 11 13 6 4 4 8 7 11 14 12 0 5 9 13 15 15 4 8 0 9 1 4 11 7 13 1 4 9 7

5 14 13 4 15 4 9 3 3 11 2 0 2 13 10 11 8 1 2 14 12 5 2 12 5 15 1 12 13 8 0 14 14 9 1 3 15 14 11 15 13 2 13 4 8 8 14 9 5 5 3 7 14 10 3 4 2 10 12 8 9 3 1 1 10 8 4 13 15 6 15 10 1 6 7 10 2 5 10 5 7 2 12 7 7 15 11 4 5 4 1 9 14 5 12 7 0 15 2 5 2 2 13 8 12 7 1 14 9 7 15 12 13 15 10 4 9 10 9 13 3 1 4 13 15 2 14 10 8 15 15 8 13 11 1 0 8 1 0 2 7 13 8 7 4 12 9 15 5 3 1]

Instance: `u_i_lohi.0`, makespan: **102136.0** Solution: [10 11 7 11 14 0 13 10 8 9 10 13 0 5 5 4 5 10 13 5 3 1 8 11 0 4 10 5 4 11 0 0 5 4 2 7 8 4 11 4 6 3 11 6 15 4 3 2 2 12 12 12 8 11 2 10 2 4 1 6 8 12 3 12 4 4 11 13 14 15 3 10 8 10 11 5 6 1 2 4 8 12 10 1 2 3 12 7 7 10 3 8 8 13 5 3 15 1 14 5 6 11 14 0 10 7 1 10 7 11 5 12 8 11 8 8 10 1 5 6 0 3 2 0 5 0 5 15 3 2 13 11 14 4 1 11 6 11 13 2 0 0 13 4 14 9 1 1 13 6 1 2 15 10 11 14 0 15 11 0 4 7 14 12 13 1 6 14 10 14 14 6 15 11 9 8 1 15 8 10 15 14 12 12 13 9 10 7 6 10 15 3 15 6 5 14 14 6 8 8 3 0 2 10 8 2 12 12 7 5 7 7 15 11 14 12 12 10 13 13 1 15 8 5 10 0 12 15 1 15 3 12 9 2 0 6 15 3 2 4 1 5 10 4 12 2 11 6 0 4 8 11 15 13 4 7 0 2 9 4 9 4 8 5 4 14 3 9 0 1 14 3 2 14 4 3 8 3 14 9 14 15 2 8 10 15 6 14 7 9 15 3 10 11 0 2 11 8 1 14 7 14 4 14 15 6 10 4 8 14 1 11 12 7 13 3 2 11 7 10 6 9 13 1 6 1 11 0 6 3 14 14 11 5 13 3 10 6 2 15 8 15 7 15 14 11 0 9 0 14 11 13 8 13 9 4 12 15 13 7 8 8 8 15 1 6 13 8 14 11 2 9 14 1 15 4 7 4 9 0 6 3 10 9 4 5 10 14 5 11 7 3 1 12 11 15 4 10 2 15 9 15 10 12 6 8 7 11 0 10 12 14 7 2 15 0 13 2 15 6 6 7 10 9 13 13 11 8 6 2 3 9 5 4 12 13 4 7 8 8 3 11 13 9 14 7 5 10 15 1 9 11 6 9 14 1 13 4 1 5 13 4 11 12 1 15 0 0 6 14 2 9 1 15 11 14 4 1 12 11 9 12 7 4 14 3 0 2 13 1 12 14 14 10 8 6 14 7 5 10 4 8 12 10 4 6 0 2 4 14 3 0]

Instance: `u_i_lolo.0`, makespan: **2549.8** Solution: [4 12 7 14 7 6 9 1 13 14 9 14 3 8 3 12 4 6 10 15 2 5 7 8 2 4 4 0 1 3 6 0 6 10 9 12 15 6 4 1 5 1 13 3 13 5 13 11 6 0 6 11 1 4 8 9 12 5 6 4 6 5 4 8 7 2 10 9 5 11 10 14 11 7 0 14 9 5 2 9 14 6 12 4 13 13 1 9 11 15 12 5 5 15 2 2 15 1 15 5 15 4 10 0 3 2 8 15 1 10 0 11 11 9 12 13 2 7 0 14 15 8 10 15 4 3 0 7 14 1 8 6 7 3 13 0 1 6 4 2 15 3 9 13 1 7 10 5 5 5 3 3 4 1 14 7 11 6 12 2 2 9 12 10 12 11 6 8 8 10 9 4 0 2 9 6 2 7 10 0 0 2 12 2 2 15 12 3 6 4 3 7 15 12 11 0 15 4 9 4 10 7 13 15 0 11 11 8 8 1 14 4 1 10 6 4 13 11 15 13 4 0 12 1 13 6 8 7 1 15 0 9 9 8 4 0 11 3 15 10 8 10 14 10 10 15 15 1 7 8 5 6 13 4 3 3 14 15 4 13 11 11 3 7 1 8 10 3 2 6 15 7 15 9 0 14 1 2 8 3 0 10 5 13 6 15 13 14 9 3 7 12 3 5 0 10 7 8 7 14 12 11 1 7 8 2 0 12 11 14 14 4 4 12 7 9 8 3 15 3 10 1 10 12 11 12 6 8 2 10 10 2 12 7 7 5 12 9 14 11 13 5 8 1 1 5 11 15 4 3 11 15 6 14 13 4 6 10 7 4 7 14 9 0 12 6 3 12 3 15 2 2 15 5 8 9 14 12 1 10 9 11 13 1 6 5 6 2 3 8 7 5 4 13 8 8 15 7 14 12 2 4 6 5 11 9 1 15 15 7 13 6 10 7 10 2 4 7 4 13 9 13 2 13 1 9 2 11 13 5 10 13 3 1 3 13 0 13 14 0 11 12 5 6 13 7 1 7 10 15 13 3 2 9 0 11 4 9 0 14 0 1 3 14 1 7 2 1 0 4 0 9 12 5 14 10 5 3 1 12 5 0 14 7 4 2 0 2 6 9 9 9 1 10 11 2 3 4 11 0 8 11 1 3 3 11 12 10 6 14 13 7]

Instance: `u_s_lohi.0`, makespan: **123251.5** Solution: [4 7 0 0 11 7 9 2 0 2 1 14 3 9 9 0 9 13 4 3 2 4 9 3 12 1 5 2 0 0 11 11 0 2 7 15 0 7 0 8 13 2 10 0 3 2 0 4 1 8 13 1 1 9 4 3 1 3 9 4 3 0 4 5 5 9 13 7 14 0 2 1 0 3 13 13 1 2 11 0 5 5 11 2 11 3 0 4 13 0 7 13 7 12 3 2 5 1 5 9 15 12 9 4 7 3 3 13 13 0 15 0 3 3 5 0 2 15 9 5 0 5 2 0 0 15 11 15 5 0 11 1 9 11 0 9 0 15 13 0 15 1 13 0 2 11 4 11 5 13 2 15 2 1 1 5 5 13 0 0 1 4 9 9 15 2 15 11 5 5 9 13 13 9 9 3 0 7 15 15 13 2 2 13 7 8 11 11 0 7 0 0 1 0 13 0 2 2 13 0 4 1 1 13 0 0 14 1 0 15 0 0 8 0 0 0 0 5 6 5 4 1 2 9 8 7 2 0 5 5 9 10 0 15 7 15 9 4 15 0 4 11 3 3 4 6 9 9 2 10 1 15 3 7 2 0 0 4 13 0 1 4 3 7 9 2 8 9 11 4 5 13 0 1 11 2 8 13 13 7 1 11 2 3 5 1 0 11 0 14 15 1 6 15 7 7 8 0 5 0 0 0 3 0 11 1 1 10 11 3 0 9 0 0 9 15 11 1 9 0 9 11 7 4 0 11 6 7 4 3 9 5 2 0 11 13 0 5 1 13 15 0 0 0 0 13 0 11 0 0 11 15 0 13 11 11 4 0 0 7 5 9 15 14 0 15 10 6 13 7 6 6 1 1 7 7 13 0 13 13 1 5 5 0 0 11 9 14 13 4 0 13 3 13 5 13 4 1 7 8 9 5 6 1 15 9 7 9 15 5 11 9 13 0 15 3 13 0 5 0 13 5 13 0 7 2 9 7 11 5 7 14 5 15 0 3 3 0 1 11 14 11 15 13 15 12 9 4 2 13 0 0 0 15 13 0 2 1 4 0 9 13 9 7 3 7 13 9 0 11 14 9 13 3 0 2 3 7 9 13 2 15 2 1 13 13 9 0 1 9 7 5 15 1 6 4 0 15 1 4 1 5 15 15 0 0 0 9 11 0 2 0]

Instance: `u_s_lolo.0`, makespan: **3450.1** Solution: [6 5 6 8 10 15 1 15 6 0 1 13 11 9 2 4 5 4 9 9 9 5 7 5 6 3 1 0 13 2 15 4 11 14 15 3 9 7 10 15 7 9 7 3 11 7 9 12 2 3 0 4 15 15 12 12 1 3 7 10 7 2 4 11 4 11 15 3 9 0 12 4 0 4 3 13 2 3 6 4 1 7 13 2 5 0 15 1 5 13 9 7 1 15 14 1 6 3 13 1 13 4 9 9 7 0 0 12 14 4 11 15 10 5 3 9 13 15 5 9 1 11 7 3 4 7 1 13 11 7 12 1 15 0 11 2 8 15 13 11 13 1 5 15 3 9 7 6 0 0 3 9 1 15 6 0 0 15 3 5 15 13 11 7 9 0 0 1 5 9 0 0 1 13 2 8 15 5 3 1 3 5 2 2 5 15 13 15 1 3 0 9 2 13 4 4 11 15 6 11 4 3 9 15 5 2 9 6 11 1 13 11 3 2 3 7 11 15 9 1 15 14 1

4 5 2 13 11 6 9 5 7 2 7 1 5 9 7 3 11 13 9 8 7 6 9 0 3 15 3 9 4 4 15 7 14 2 11 1 0 12 9 11 13 13 11 2 4 2 12 2 8 1 3 11 7 14 0 6 10 1 9 15 0 11 15 5 2 7 7 11 1 3 10 1 0 5 12 7 2 3 15 1 2 5 7 0 8 8 7 7 13 2 2 6 0 5 13 3 4 12 0 11 1 0 11 9 9 13 3 13 0 13 2 15 5 11 0 8 9 14 2 10 11 5 5 10 5 9 6 7 2 11 15 2 1 12 0 13 2 8 2 7 13 11 3 7 1 0 5 3 10 1 11 2 7 0 0 2 9 7 6 15 14 6 0 11 5 7 11 13 5 13 11 12 5 13 13 6 13 3 7 5 9 13 5 8 15 6 1 9 11 0 15 8 3 5 7 1 6 8 15 6 4 4 13 9 1 8 4 7 14 1 1 1 0 0 11 9 8 2 0 1 0 0 4 9 0 0 0 3 13 2 13 6 15 13 8 4 7 1 11 11 14 14 1 2 10 0 9 14 0 7 7 13 11 5 4 10 15 2 5 7 15 0 11 3 10 13 0 1 0 7 7 0 2 11 11 8 0 7 0 4 5 3 0 10 13 3 11 5 15]

pμ-CHC

Instance: `u_c_hihi.0`, makespan: **7381570.0** Solution: [6 0 0 0 0 0 0 3 1 0 0 0 8 6 6 0 0 0 0 2 0 13 3 3 0 10 0 0 10 4 0 0 0 1 4 0 6 5 6 4 0 1 6 1 7 11 4 7 1 0 12 0 1 0 0 2 3 3 15 3 3 5 6 8 0 0 0 0 7 0 1 0 14 3 1 0 0 0 2 0 0 13 8 0 0 0 5 0 4 1 0 0 0 0 3 0 0 0 4 0 2 0 0 1 1 0 13 0 0 0 0 0 0 3 1 0 0 7 2 9 2 1 0 0 0 6 0 1 2 1 0 8 1 15 3 0 0 0 5 2 15 5 11 0 9 9 0 0 7 10 1 2 0 0 14 0 11 0 1 2 15 0 9 1 1 0 2 10 0 0 5 2 3 1 1 1 0 3 13 10 6 1 0 0 4 0 0 0 0 0 5 7 0 2 2 3 0 3 0 2 9 0 0 0 5 1 0 13 2 1 0 2 2 8 0 1 0 4 10 0 0 2 1 0 0 1 5 0 11 9 5 0 7 0 0 0 13 0 0 7 1 0 0 0 10 5 0 0 4 0 11 4 7 10 0 3 0 0 1 2 0 6 0 1 1 0 14 3 7 12 0 0 1 1 3 0 0 0 0 15 1 0 0 0 14 0 2 12 0 9 2 13 1 0 0 1 0 1 0 5 0 0 0 0 0 8 1 0 14 11 5 0 8 1 2 0 0 0 1 0 0 0 0 1 8 1 0 13 0 0 0 8 0 0 0 10 14 2 0 1 1 0 7 1 9 5 13 0 5 0 4 0 1 0 0 0 0 4 3 1 0 6 4 2 0 0 4 1 5 1 2 10 1 0 0 8 1 2 1 1 0 0 0 3 0 2 0 0 1 0 2 12 2 0 0 2 10 0 8 3 0 8 0 0 1 1 0 3 12 0 0 2 9 3 7 0 2 3 5 2 9 0 10 2 0 0 0 1 0 0 0 2 0 1 10 1 6 4 1 2 0 0 1 0 0 9 0 0 12 0 3 9 0 11 0 1 0 3 3 1 0 0 6 1 6 0 0 8 0 0 0 2 0 0 2 1 1 7 0 1 0 1 3 0 2 14 1 0 1 7 6 2 13 3 3 0 1 0 4 0 8 0 0 6 12 0 0 0 0 1 9 4]

Instance: `u_c_hilo.0`, makespan: **153105.4** Solution: [11 10 0 9 5 1 12 13 3 9 4 4 6 0 13 0 0 2 1 1 2 4 0 3 1 2 4 11 0 15 14 7 5 1 1 3 0 5 0 3 7 8 9 2 7 0 0 13 10 0 0 15 0 5 5 1 3 2 0 0 0 3 2 2 1 11 0 0 8 1 0 3 0 10 11 4 7 13 5 13 1 2 10 14 0 0 4 0 1 6 0 2 9 9 4 8 0 0 9 3 0 1 11 15 6 2 4 2 7 11 10 7 7 3 9 11 4 4 0 1 7 4 9 0 4 5 0 0 0 1 0 1 3 2 0 1 10 15 6 6 2 4 0 9 7 15 1 10 12 5 8 0 6 9 2 5 5 7 1 2 7 1 4 1 5 2 5 0 2 1 6 15 5 5 8 1 1 4 1 0 11 9 4 1 6 3 3 10 0 4 10 13 8 6 2 13 1 1 1 1 6 6 3 1 11 1 3 0 1 1 3 12 1 4 8 10 12 8 9 0 0 1 12 2 9 3 0 0 2 6 0 3 2 0 4 1 0 1 7 1 1 10 2 12 14 3 1 4 0 0 1 8 0 3 10 1 0 15 1 2 5 0 9 2 14 2 13 2 10 6 8 13 0 0 6 2 1 0 11 8 9 3 1 1 6 9 2 1 8 2 11 15 3 4 0 0 0 1 7 8 6 2 0 7 0 7 2 10 1 0 2 7 1 2 3 2 7 2 1 2 3 12 4 3 6 1 4 0 1 8 4 3 1 0 13 10 2 0 4 5 1 6 10 6 2 0 1 9 15 12 6 0 7 13 3 7 1 5 1 5 3 11 5 7 7 12 3 0 0 1 6 15 4 2 8 0 14 1 3 9 0 5 11 2 4 4 0 3 3 7 2 7 15 3 9 12 5 14 9 4 5 0 2 0 0 13 5 5 0 8 3 5 12 3 9 1 10 4 2 1 2 1 5 5 5 8 15 8 0 14 2 4 0 2 3 6 3 0 0 13 6 2 4 3 6 6 2 5 2 1 6 4 0 8 6 6 14 6 6 2 1 0 0 1 11 15 5 1 11 2 1 0 12 1 0 0 3 12 0 3 1 2 5 4 1 1 0 11 4 12 8 2 1 0 10 3 9 0 11 6 0 9 3 12 5 3 8 14 4 3 3 3]

Instance: `u_c_lohi.0`, makespan: **239260.0** Solution: [0 0 1 0 8 11 1 0 0 6 2 9 1 1 15 2 0 0 0 0 3 9 0 1 4 0 1 0 3 14 0 0 9 15 4 0 0 0 0 6 0 0 0 13 6 8 1 0 3 2 0 0 7 1 1 0 1 4 3 0 2 1 6 11 1 1 1 1 0 1 0 2 7 5 0 6 0 0 1 0 0 0 0 0 6 0 1 0 2 0 0 1 0 0 1 10 14 1 2 15 0 5 1 11 0 1 1 1 0 11 1 1 1 2 0 0 0 4 1 0 4 0 0 0 1 6 7 1 4 8 4 0 0 0 1 0 3 0 8 13 1 0 0 0 7 2 0 5 0 4 0 2 2 1 1 0 0 0 1 1 4 5 8 12 0 0 0 5 3 0 0 0 5 9 2 3 0 0 0 0 1 14 0 5 3 6 5 0 3 3 0 0 0 0 1 0 5 0 1 3 0 12 2 3 0 3 1 1 0 0 2 0 5 0 11 0 7 0 0 2 0 11 0 6 0 0 0 0 5 2 0 0 0 0 1 0 10 13 1 8 8 0 13 0 0 0 0 7 0 3 2 2 3 0 4 3 0 2 1 10 14 9 0 11 0 0 4 1 0 6 1 0 0 2 1 12 0 0 0 0 0 2 1 4 8 0 12 0 5 5 4 8 0 0 0 0 2 4 7 0 7 0 0 0 9 1 0 3 0 2 1 0 1 1 1 0 3 0 9 15 1 5 0 0 2 0 0 0 5 7 1 1 2 0 4 1 0 0 0 2 2 0 2 0 0 0 0 7 2 3 0 15 3 0 1 0 0 3 1 0 0 13 0 0 3 0 0 0 4 1 0 0 3 0 0 0 0 0 0 2 0 12 0 0 1 1 0 0 1 10 1 1 2 2 2 0 0 1 0 8 4 1 12 10 0 0 4 0 0 6 2 0 2 4 0 0 0 6 4 0 0 4 0 8 6 0 0 5 3 2 0 1 2 0 0 12 0 0 13 2 1 0 0 0 0 1 15 1 1 0 7 0 0 8 0 0 7 0 13 6 0 0 11 2 0 1 1 0 0 3 1 0 0 0 0 0 8 3 2 5 1 1 0 0 2 7 0 9 5 0 2 9 0 0 3 0 9 0 0 14 7 2 3 0 0 4 0 0 0 0 2 0]

Instance: `u_c_lolo.0`, makespan: **5147.9** Solution: [4 6 1 0 6 9 5 6 10 3 3 1 5 1 13 2 4 5 0 6 5 0 3 0 0 5 4 14 2 4 11 12 0 2 6 3 13 1 5 1 3 5 2 10 9 0 6 1 4 0 1 8 7 6 0 13 12 11 10 0 0 7 4 4 1 7 0 1 2 0 2 2 1 4 15 1 0 0 2 0 10 2 4 13 2 1 2 6 3 9 6 5 1 3 3 0 2 10 5 1 15 8 8 10 5 0 1 0 3 1 7 8 15 2 2 12 5 0 2 15 6 2 3 5 8 5 14 2 3 7 3 7 3 12 14 1 5 9 3 0 8 4 0 10 11 11 12 7 14 6 1 1 4 1 0 7 3 9 12 12 0 6 1 0 11 0 2 0 5 3 1 10 4 4 1 0 3 1 9 0 1 0 1 9 9 1 8 2 1 15 0 6 8 12 14 0 2 4 14 0 2 7 12 3 2 7 4 0 2 13 0 4 4 2 5 14 5 0 14 1 3 5 0 15 4 6 1 6 2 14 0 5 11 0 10 14 0 3 15 7 4 9 0 2 2 9 6 7 8 3 7 7 11 6 1 13 2 1 4 0 12 2 0 11 13 1 4 0 9 9 11 5 8 7 3 6 2 8 10 0 6 2 2 10 12 4 8 11 3 0 8 2 0 0 1 4

1 13 11 1 1 0 9 14 0 7 11 0 1 0 8 2 1 10 9 0 3 10 7 3 0 4 3 6 4 0 1 3 3 0 1 3 2 1 2 3 4 2 1 14 3 8 8 2 14 5 3 8 15 10 7 10 0 7 2 11 0 1 7 1 5 14 2 0 4 0 0 12 10 14 2 4 1 4 11 4 1 4 2 0 2 0 10 2 0 0 1 3 1 3 10 0 0 15 4 2 6 3 3 1 12 6 2 4 6 14 5 3 9 15 11 5 15 8 4 2 2 3 1 1 5 0 13 0 7 9 2 11 1 11 5 2 6 4 5 3 4 13 1 1 7 4 7 15 3 3 4 5 0 5 2 9 8 0 6 1 1 9 13 5 0 5 7 3 9 2 4 4 1 6 9 0 3 14 3 9 3 6 4 1 5 4 11 14 10 1 0 0 0 2 7 0 10 13 11 3 6 0 7 1 14 0 6 1 11 7 8 0 0 15 9 8]

Instance: `u_i_hihi.0`, makespan: **2938380.8** Solution: [9 10 11 11 2 7 6 7 7 15 2 2 15 9 1 8 15 15 0 13 14 10 4 5 1 8 13 3 11 3 5 4 10 6 6 14 1 11 14 4 10 12 6 8 0 9 0 7 0 9 2 3 0 9 1 8 11 5 2 4 14 13 11 2 9 8 7 15 12 6 5 7 0 8 3 10 12 11 7 2 2 7 0 14 10 0 9 1 10 14 2 3 13 12 7 2 4 11 8 14 1 8 10 4 8 0 10 2 5 14 12 10 0 9 15 13 13 4 14 12 8 10 5 1 4 6 0 11 0 4 13 10 9 14 1 6 4 1 15 5 15 6 1 11 5 12 1 2 5 10 5 1 6 2 3 5 6 7 3 14 7 2 4 2 8 6 4 5 2 5 11 6 9 10 1 13 3 8 2 15 5 9 15 13 8 1 1 14 9 9 3 0 4 14 5 0 12 2 15 8 9 6 12 8 0 13 10 0 4 4 5 2 6 2 6 15 2 3 12 14 4 5 11 7 15 12 13 15 12 11 15 13 6 7 13 14 4 15 13 8 11 15 10 6 4 8 15 14 1 10 12 5 12 4 3 2 10 3 6 6 2 11 7 5 5 3 4 7 0 13 0 8 6 1 0 12 7 9 10 11 2 15 6 7 9 6 6 3 5 15 6 3 11 13 7 7 0 5 2 4 0 1 15 14 2 14 9 14 14 8 13 1 4 13 8 6 3 11 9 11 6 12 9 10 4 4 11 13 13 1 9 9 5 12 10 2 15 8 14 14 1 3 0 12 11 5 15 5 5 14 5 3 1 0 6 13 8 6 12 3 8 10 12 6 9 2 9 3 6 1 11 12 6 9 11 8 13 15 12 6 0 14 8 7 5 8 12 14 4 0 5 14 6 7 2 14 15 0 5 14 15 8 10 13 1 8 10 9 7 6 14 4 14 8 13 12 14 10 9 4 5 11 0 0 4 8 11 11 14 4 14 14 10 9 11 10 3 8 7 14 11 5 12 5 14 1 2 0 2 15 5 1 3 5 3 12 3 14 3 7 3 12 12 2 8 2 12 6 1 9 5 11 10 12 11 4 6 11 9 3 5 0 4 12 13 7 11 3 12 5 11 8 15 11 13 9 6 0 4 6 8 2 9 12 10 14 14 13 1 5 2 9]

Instance: `u_i_hilo.0`, makespan: **73378.0** Solution: [4 1 0 13 5 8 14 10 13 13 0 4 1 11 6 14 15 5 13 0 2 1 11 14 3 4 12 4 7 15 1 5 12 12 10 10 4 12 8 8 12 11 3 11 5 10 5 8 6 4 3 8 14 14 6 0 14 2 9 12 7 8 9 11 8 8 2 3 9 8 4 4 15 5 3 9 14 12 13 2 5 3 11 2 8 14 12 10 1 11 0 8 7 5 9 7 2 4 14 15 13 1 9 4 6 15 11 14 1 8 4 12 1 8 6 0 1 7 9 7 12 8 1 2 8 10 7 8 4 2 11 14 1 0 14 7 12 4 6 4 10 2 3 1 7 3 0 14 11 1 12 11 2 15 3 2 10 8 1 0 0 14 2 1 1 15 0 3 5 1 6 1 11 0 11 10 1 0 6 12 6 0 12 2 6 11 0 2 9 6 2 7 0 3 1 0 3 1 11 11 15 8 4 13 11 6 0 0 6 12 6 12 3 9 3 10 12 9 15 6 7 9 10 13 14 9 0 5 3 15 0 11 0 3 0 11 7 11 6 15 6 5 9 10 2 7 10 3 13 9 15 6 0 0 5 13 12 13 10 14 8 9 1 10 0 1 0 3 10 13 15 7 13 12 6 7 12 1 2 15 10 12 6 5 13 10 3 5 0 13 15 3 12 1 14 11 1 14 12 14 11 3 12 3 6 9 4 0 15 11 1 6 13 3 11 12 6 14 9 5 8 3 10 8 5 0 6 8 1 1 0 9 11 10 6 1 4 8 7 11 5 12 3 5 9 13 14 15 4 9 0 9 1 4 15 7 7 1 6 9 7 14 14 13 4 15 4 10 13 4 11 2 10 7 13 10 13 8 1 2 5 12 2 2 12 5 15 1 12 13 8 0 14 14 9 1 3 15 14 11 15 13 2 13 4 8 8 4 9 5 5 3 7 14 10 3 3 2 10 12 8 9 3 1 1 4 8 4 13 15 1 15 10 1 6 7 10 2 5 13 5 7 2 12 7 7 15 11 4 5 4 1 9 14 5 10 7 0 15 2 5 2 2 14 2 12 7 1 14 9 3 15 12 8 15 10 4 9 14 9 13 3 1 4 9 5 2 14 10 8 15 9 8 13 11 13 0 8 1 0 2 7 13 8 7 4 12 9 15 5 3 1]

Instance: `u_i_lohi.0`, makespan: **102050.6** Solution: [10 11 7 11 14 0 13 12 8 9 15 13 0 5 5 4 5 10 13 5 3 1 8 11 0 4 10 5 4 11 0 0 5 3 2 7 8 4 11 14 6 3 11 6 11 4 3 2 2 12 12 12 12 11 2 2 3 4 1 1 8 12 3 12 4 4 11 13 14 15 3 15 8 8 11 5 6 1 2 4 9 12 10 1 2 3 8 7 7 10 15 8 8 13 5 3 15 1 14 5 6 11 14 0 10 7 1 3 7 11 5 5 8 9 1 8 10 1 5 6 0 3 2 0 5 4 5 7 3 2 13 11 14 5 1 11 6 11 13 2 0 0 13 4 4 9 1 5 13 6 7 2 15 8 11 14 0 9 2 0 4 7 14 12 13 1 4 14 10 14 14 6 15 11 9 3 1 15 8 5 15 14 12 12 13 9 10 7 6 10 15 3 15 6 5 14 14 6 8 8 3 0 2 10 8 2 12 10 1 5 7 7 15 11 5 12 12 3 13 7 1 15 8 5 10 0 12 15 1 15 3 12 9 2 0 6 15 3 2 4 1 5 10 4 12 2 11 6 0 4 8 11 15 13 4 7 0 2 9 4 9 4 8 5 1 14 3 9 0 1 14 15 2 14 4 3 8 3 14 9 14 15 2 8 10 15 6 14 7 9 15 3 10 11 3 2 11 8 1 14 7 14 4 14 15 0 4 4 8 14 1 11 12 7 13 3 2 11 4 7 6 9 13 1 6 1 11 0 10 3 14 14 11 5 13 3 10 6 2 15 11 15 7 15 14 11 0 9 0 13 11 13 8 0 9 4 12 15 13 2 11 12 8 15 1 0 13 15 14 11 2 9 14 1 15 4 7 4 8 0 6 3 10 9 4 12 10 14 5 11 7 3 1 12 12 15 4 10 2 15 9 15 10 12 6 8 3 11 0 10 12 5 7 2 15 13 13 2 15 6 6 7 10 9 13 13 11 6 6 2 3 9 5 4 12 13 4 7 8 8 6 11 13 9 14 7 5 9 15 1 9 11 6 9 14 1 13 4 1 5 13 4 11 12 1 15 0 0 5 4 2 9 1 15 11 14 4 1 12 11 9 12 7 4 14 3 0 2 13 1 5 14 14 10 8 6 14 7 5 10 4 13 12 10 4 6 0 2 4 14 2 0]

Instance: `u_i_lolo.0`, makespan: **2541.4** Solution: [4 12 7 14 7 11 9 1 13 15 9 14 3 8 3 12 4 6 10 15 2 5 7 8 14 3 4 6 1 3 6 0 6 10 9 12 15 6 4 1 5 1 13 3 13 5 13 11 6 0 6 11 1 4 8 9 12 5 11 4 6 5 4 8 7 2 10 9 5 11 10 14 12 7 0 14 9 5 2 9 14 6 12 4 13 13 1 9 11 15 0 5 5 15 2 2 12 1 15 5 12 4 14 0 3 2 8 15 1 10 0 11 11 9 5 13 2 7 0 14 15 8 1 15 4 3 0 12 14 1 8 6 7 4 13 0 1 4 4 2 15 3 9 13 3 7 10 5 6 5 3 3 4 1 14 7 15 6 12 2 2 14 12 6 2 11 6 13 8 10 9 3 0 2 9 6 2 7 10 0 0 2 12 2 2 15 13 3 6 4 3 7 15 1 11 0 15 4 9 4 10 7 13 15 0 11 11 8 8 10 14 4 1 10 6 4 13 11 15 13

4 0 12 1 13 9 8 7 1 15 0 9 9 8 1 0 11 3 15 11 8 8 14 10 10 15 15 1 7 8 5 6 13 4 3 15 14 15 4 4 11 12 3 7 1 0 10 3 2 6 15 11 15 9 0 14 1 2 8 3 0 0 5 13 6 15 14 14 5 3 7 12 3 5 0 10 7 8 7 3 12 11 1 11 8 2 0 12 11 14 14 4 4 12 7 7 8 3 14 3 8 5 10 12 11 12 6 8 2 10 10 6 12 7 7 5 12 9 14 11 13 5 8 1 1 5 11 15 4 3 11 15 12 12 13 4 6 6 7 8 7 14 9 0 12 6 3 12 3 8 2 2 15 5 8 9 14 12 1 10 9 11 12 1 6 5 6 2 3 10 7 5 4 13 8 8 10 7 14 12 5 4 6 5 11 9 14 12 14 7 13 6 9 3 10 2 11 7 4 13 9 13 2 13 1 9 2 11 13 5 10 13 3 1 3 13 0 13 5 2 11 12 5 6 13 7 3 7 10 15 13 3 2 9 15 11 4 9 0 6 0 1 3 14 1 7 10 6 0 8 0 9 12 5 14 10 5 5 1 12 5 0 14 7 10 14 0 2 1 9 9 9 1 10 11 2 5 4 9 0 8 11 9 10 3 11 12 10 6 14 13 7]

Instance: `u_s_hihi.0`, makespan: **4103500.3** Solution: [0 13 0 12 3 2 9 11 10 3 15 0 0 3 0 13 4 9 0 11 3 0 11 0 0 0 2 7 15 2 13 15 0 0 5 2 2 11 2 2 0 8 3 13 0 0 2 9 0 2 13 0 3 2 6 1 5 7 2 1 1 0 3 7 13 9 6 15 13 1 0 13 7 15 1 0 11 0 3 6 3 7 1 9 0 11 9 1 15 5 11 3 5 13 0 15 7 7 0 11 9 5 11 4 3 13 0 0 4 7 0 2 15 5 2 1 9 5 0 11 9 0 9 0 1 9 13 0 9 15 5 13 7 0 5 5 11 3 2 15 9 5 11 1 9 2 2 0 9 15 13 1 5 10 0 11 11 13 1 9 0 11 0 11 1 0 0 3 2 5 7 4 9 4 13 0 0 0 0 7 9 0 15 9 13 3 13 0 0 1 4 0 3 11 13 0 9 15 0 12 5 3 9 4 0 1 11 0 5 0 4 15 1 0 3 15 5 3 15 7 2 9 2 15 11 13 1 13 7 11 5 11 2 9 9 15 0 13 0 11 1 7 7 1 0 5 1 14 5 5 1 2 13 9 6 3 13 5 1 11 0 11 0 2 5 3 8 2 13 0 13 7 0 15 5 7 5 12 5 6 0 0 13 13 15 6 9 3 7 5 0 1 10 0 1 4 4 3 0 4 2 6 4 0 15 0 0 8 9 5 15 3 14 1 0 2 1 0 9 4 0 7 13 0 0 13 3 8 2 15 3 7 5 13 4 10 2 15 0 7 6 13 4 0 11 3 9 0 3 7 7 0 0 3 3 1 1 0 13 13 9 7 7 13 4 7 4 0 2 3 0 3 0 11 0 11 5 1 3 9 15 7 11 12 5 2 3 4 3 5 3 0 15 4 13 0 15 9 9 2 3 10 7 2 11 14 2 3 3 0 10 0 9 9 0 15 7 5 3 1 4 3 7 0 11 0 5 9 8 9 5 0 9 2 7 11 0 0 1 9 3 7 5 11 3 7 0 0 7 7 13 9 3 13 7 7 0 4 5 0 11 15 1 13 9 11 7 12 2 15 4 0 3 3 11 0 7 13 11 0 3 14 8 0 1 7 7 0 1 11 0 9 0 5 13 1 15 0 12 2 5 13 15 5 9 2 12 0 0 6 7 5]

Instance: `u_s_hilo.0`, makespan: **95787.4** Solution: [4 7 10 4 1 5 5 6 3 15 2 11 15 0 2 1 5 5 12 11 6 3 15 12 13 0 15 13 9 3 6 6 0 9 3 4 4 7 15 3 3 8 13 2 10 11 1 7 4 2 3 2 6 6 11 5 4 0 7 5 7 3 1 6 9 13 12 6 9 12 0 2 1 5 12 13 9 15 2 3 3 6 7 11 4 1 15 8 15 15 15 9 5 4 5 8 1 13 1 2 0 1 11 9 4 9 9 9 13 4 15 2 10 7 1 5 5 15 4 2 12 2 1 9 13 13 11 7 7 13 11 3 1 9 0 12 5 0 5 2 1 5 2 11 3 2 0 14 14 3 6 1 1 5 2 7 5 9 13 11 11 13 3 1 15 7 5 5 3 13 5 5 0 9 9 3 11 13 4 2 14 9 13 1 1 13 2 4 1 14 13 3 7 4 0 13 11 2 4 14 2 2 5 3 3 4 1 13 13 1 13 5 15 1 15 3 2 0 4 0 10 5 8 1 11 0 4 3 4 3 11 0 8 0 15 0 15 7 4 7 7 11 15 7 2 9 13 1 15 0 3 6 9 0 3 11 9 13 9 5 15 12 15 2 7 6 10 0 9 7 2 5 8 15 11 6 3 2 15 9 14 9 15 5 8 6 2 8 9 3 0 10 4 0 13 4 2 15 6 8 5 7 15 9 2 5 9 2 10 2 1 1 15 2 7 13 4 5 13 3 6 4 0 2 7 6 1 13 4 9 1 14 7 15 5 15 0 9 4 6 7 9 8 3 10 13 2 0 12 11 9 3 13 7 11 7 13 4 14 7 5 3 14 1 1 3 15 10 13 7 0 7 7 1 4 7 0 5 5 6 7 0 8 15 2 0 15 2 0 0 8 9 4 11 0 15 11 5 6 0 3 15 7 10 5 13 11 8 11 13 4 5 0 5 1 12 11 11 15 13 7 0 13 5 14 13 13 0 3 0 6 7 0 13 6 0 1 3 11 15 6 3 1 13 13 1 9 2 9 11 0 10 0 2 0 12 15 7 14 0 5 1 0 5 5 13 3 13 15 13 11 0 11 5 13 8 11 11 11 0 10 3 1 11 9 9 2 9 1 8 11 9 1 9 3 0 9 5 11 0 5 3 5 0 12 11 0 7 15 10 7 7]

Instance: `u_s_lohi.0`, makespan: **122083.3** Solution: [15 7 0 0 11 7 9 2 0 2 1 4 3 9 1 0 9 13 4 3 2 4 15 3 3 1 7 2 0 0 5 11 0 2 7 15 0 7 0 8 13 10 5 0 3 2 0 4 11 8 13 1 1 9 2 5 1 3 9 4 1 0 4 5 5 0 13 7 7 3 2 1 0 3 13 13 1 2 11 0 5 5 11 2 11 3 0 4 13 0 7 13 7 12 3 2 5 1 5 9 15 2 9 2 7 5 3 13 14 0 15 0 3 3 5 0 2 15 9 5 3 5 2 0 2 9 11 15 5 0 11 1 9 11 0 9 0 15 13 0 15 1 13 7 2 11 4 11 11 13 2 15 2 1 5 5 5 13 0 0 1 4 0 9 15 2 15 11 5 5 9 13 13 9 9 3 0 7 15 15 13 2 2 15 0 8 11 2 0 7 0 0 1 0 13 0 2 2 6 0 4 1 3 13 0 2 2 1 0 15 0 4 2 0 0 0 0 5 0 0 2 1 2 9 0 7 2 0 5 5 9 12 0 15 7 15 9 4 15 0 14 15 15 3 2 6 0 0 2 12 1 15 3 7 10 15 0 4 9 0 1 4 3 7 9 3 8 9 11 4 5 13 0 1 11 2 10 13 13 7 1 11 2 3 2 1 0 11 0 5 15 1 2 15 7 7 8 0 14 0 0 0 3 0 11 1 1 10 11 3 0 7 0 0 9 15 11 1 9 0 9 11 7 4 0 11 6 7 8 3 9 5 11 0 11 13 0 5 9 13 15 0 0 0 0 13 15 11 0 0 0 15 0 13 11 11 8 0 0 7 5 9 15 15 0 7 10 6 13 3 6 6 1 1 6 7 15 3 13 0 1 1 5 0 0 11 9 2 13 4 0 13 3 0 5 13 4 1 7 8 9 5 6 1 15 9 7 9 15 5 11 3 13 1 15 3 13 0 5 0 13 5 13 0 7 15 9 7 11 5 7 15 5 15 0 3 3 0 1 11 14 11 15 13 15 13 9 4 2 13 0 0 0 15 13 0 2 1 2 0 5 13 9 7 3 7 4 9 0 11 2 10 13 3 0 2 3 7 0 0 2 15 2 1 13 13 9 0 1 9 7 5 15 1 6 2 0 15 11 4 1 5 4 15 0 7 0 9 11 0 2 0]

Instance: `u_s_lolo.0`, makespan: **3433.5** Solution: [6 5 6 8 15 7 1 15 6 0 1 13 11 9 2 4 5 4 9 9 9 5 7 5 6 3 1 7 3 2 15 4 11 1 15 13 9 15 10 15 7 9 7 3 11 7 9 8 2 3 0 4 15 15 5 12 1 3 7 8 7 2 4 11 9 11 15 3 9 8 12 4 0 3 3 13 9 3 8 4 1 7 13 2 5 0 15 1 5 13 6 7 3 15 2 1 2 3 13 1 1 4 9 9 7 0 0 12 5 4 11 15 8 5 3 9 13 15 5 9 1 11 0 3 4 7 1 13 11 7 13 1 15 0 11 11 8 2 13 11 10 1 5 0 3 9 4 12 0 2 3 0 1 15 2 0 13 15 3 5 15 13 11 7 9 5 0 1 5 9 1 0 1 13 2 8

15 5 3 5 3 5 2 2 3 15 13 15 1 3 0 9 7 14 4 4 11 15 4 11 4 3 9 15 5 2 7 6 11 1 13 11 3 2 3 7 11 15 9 1 15 12 1 4 5 2
13 0 10 9 5 7 3 7 1 1 9 7 9 11 13 9 8 7 6 9 0 3 15 3 9 4 8 2 7 14 2 11 1 0 10 9 11 13 13 11 2 4 2 14 4 14 1 3 6 13 2
0 6 10 1 9 15 0 11 15 5 2 7 7 11 1 3 2 15 0 5 12 7 2 3 4 1 2 5 7 0 8 8 7 7 13 2 2 6 0 5 13 3 4 12 0 11 1 0 11 9 9 13
3 13 0 13 2 15 5 11 0 8 9 14 2 10 11 5 5 14 5 9 6 7 2 11 3 2 1 12 1 13 2 8 2 7 13 11 3 7 1 0 12 3 10 1 11 2 7 0 0 9
9 7 6 15 13 6 0 15 5 7 11 13 5 13 11 15 5 13 13 6 13 3 7 5 9 13 5 10 15 6 1 13 11 10 15 10 3 5 7 1 6 10 15 0 4 4 14
9 1 8 4 7 10 1 5 2 0 0 11 9 8 2 0 1 0 0 4 9 0 0 0 3 13 2 13 6 15 13 8 4 7 7 11 3 5 1 1 2 10 0 9 14 0 7 3 13 11 5 4 10
15 5 2 7 15 0 11 12 8 0 3 1 0 7 7 0 2 11 11 8 9 7 6 4 5 3 0 5 13 11 11 5 15]

Appendix C

Related publications by the author

This appendix briefly comments related publications by the author and their relevance within the research reported in this thesis.

- **Evolución en el diseño y clasificación de algoritmos genéticos paralelos** (Nesmachnow, 2002), published in Actas de la XXVIII Conferencia Latinoamericana de Informática, Montevideo, Uruguay, 2002 (text in Spanish).

  The article presented a complete review of parallel models of genetic algorithms, and a historical survey of the taxonomic classifications proposed by the research community. Each parallel genetic algorithm proposal was classified into a comprehensive categorization.

- **Una versión paralela del algoritmo evolutivo para optimizacion multiobjetivo NSGA-II** (Nesmachnow, 2004), published in Actas del X Congreso Argentino de Ciencias de Computación, La Matanza, Argentina, 2004 (text in Spanish).

  The article presented the parallel version of the NSGA-II multiobjective evolutionary algorithm. The implementation was validated by applying the method to solve 18 well-known benchmark problems in multiobjective optimization, achieving accurate results.

- **Un algoritmo evolutivo multiobjetivo paralelo aplicado al diseño de redes de comunicaciones confiables** (Nesmachnow, 2005), publicado en Actas del IV Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, Granada, España, 2005, (text in Spanish).

  In this article, the previously designed parallel version of NSGA-II was applied to solve a hard combinatorial optimization problem that models the design of reliable communication networks. Accurate and efficient results were obtained for medium-sized problem instances.

- **Evolutionary Algorithms Applied to Reliable Network Communication Design** (Nesmachnow et al., 2007), published in Engineering Optimization 39 (7), 2007.

This work applied sequential and parallel evolutionary algorithms to the Generalized Steiner Problem, an NP-hard optimization problem arising in network design. A parallel version of CHC achieved the best results in terms of solution quality and computational efficiency.

- **Algoritmos evolutivos paralelos para despacho de tareas en entornos heterogéneos** (Nesmachnow, 2009), published in Actas del VI Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, Málaga, España, 2009, (text in Spanish).

  This conference article presented the first results on applying traditional evolutionary algorithms to solve the standard HCSP instances by Braun et al. (2001). The evolutionary model of CHC showed promising results when solving the low-sized HCSP instances.

- **Heterogeneous computing scheduling with evolutionary algorithms** (Nesmachnow et al., 2010), accepted for publication in Soft Computing, 2010 (to appear).

  The article presented the application of sequential and parallel evolutionary algorithms to solve the standard HCSP instances by Braun et al. (2001). The parallel version of CHC obtained the best results for the set of HCSP instances faced, and a preliminary study of the scalability of the parallel algorithm was provided.

- **Scheduling in heterogeneous computing and grid environments using a parallel CHC evolutionary algorithm**, Computational Intelligence, submitted to publication.

  The work introduced the new set of large-sized HCSP instances designed to model realistic HC and grid infrastructures. The parallel CHC method was applied to solve all the new problem instances. Accurate results were obtained when comparing with those computed by deterministic heuristics and the parallel algorithm showed a good scalability behavior when solving large HCSP instances.

- **A parallel micro-CHC evolutionary algorithm for heterogeneous computing and grid scheduling**, IEEE Transactions of Evolutionary Computation submitted to publication.

  The article presented the design and implementation of the new parallel micro CHC algorithm. The new method was applied to solve the standard HCSP instances from Braun et al. (2001) and the previously proposed large-sized HCSP instances. The results demonstrated that the parallel micro CHC algorithm is the current state-of-the-art method for the HCSP, significantly improving over the makespan results computed by deterministic heuristics and the previous parallel CHC method.

Bibliography

A. Abraham, R. Buyya, and B. Nath. Nature heuristics for scheduling jobs on computational grids. In *Proceedings of 8th IEEE International Conference on Advanced Computing and Communications*, pages 45–52, 2000.

E. Alba. *Parallel Metaheuristics: A New Class of Algorithms*. Wiley, 2005. ISBN 0471678066.

E. Alba, F. Almeida, M. Blesa, C. Cotta, M. Diaz, I. Dorta, J. Gabarró, J. González, C. León, L. Moreno, J. Petit, J. Roda, A. Rojas, and F. Xhafa. Mallba: A library of skeletons for combinatorial optimisation. In *Proceedings of the Euro-Par*, pages 927–932, 2002.

E. Alba and B. Dorronsoro. *Cellular Genetic Algorithms*, volume 42 of *Operations Research/Computer Science Interfaces*. Springer-Verlag Heidelberg, 2008. ISBN 9780387776095.

E. Alba and G. Luque. A new local search algorithm for the DNA fragment assembly problem. In C. Cotta and J. van Hemert, editors, *Proceedings of 7th European Conference on Evolutionary Computation in Combinatorial Optimization*, volume 4446 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2007.

E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, 2002.

S. Ali, H. Siegel, M. Maheswaran, S. Ali, and D. Hensgen. Task execution time modeling for heterogeneous computing systems. In *Proceedings of the 9th Heterogeneous Computing Workshop*, page 185, Washington, DC, USA, 2000. IEEE Computer Society.

T. Anderson, D. Culler, D. Patterson, and the NOW team. A case for now (networks of workstations). *IEEE Micro*, 15(1):54–64, 1995.

R. Armstrong, D. Hensgen, and T. Kidd. The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions. In *Proceedings of the 7th Heterogeneous Computing Workshop*, page 79, Washington, DC, USA, 1998. IEEE Computer Society.

T. Bäck, D. Fogel, and Z. Michalewicz, editors. *Handbook of evolutionary computation*. Oxford University Press, 1997.

H. Barada, S. M. Sait, and N. Baig. Task matching and scheduling in heterogeneous systems using simulated evolution. *Proceedings of the 10th Heterogeneous Computing Workshop, International Parallel and Distributed Processing Symposium*, 2:875–882, 2001.

F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, S. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crumme, D. Reed, L. Torczon, and R. Wolski. The GrADS project: Software support for high-level grid application development. *International Journal of High Performance Computing Applications*, 15(4):327–344, 2001.

F. Berman, G. Fox, and A. Hey. *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons, Inc., New York, NY, USA, 2003. ISBN 0470853190.

C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task scheduling strategies for workflow-based applications in grids. In *Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid*, pages 759–767, Washington, DC, USA, 2005. IEEE Computer Society.

BOINC. BOINC statistics page. URL: `http://boinc.berkeley.edu/`, consulted September 2009.

G. Borriello and D. Miles. Task scheduling for real-time multi-processor simulations. *Proceedings of the 11th IEEE Workshop on Real-Time Operating Systems and Software*, pages 70–73, 1994.

W. Bossert. Mathematical optimization: Are there abstract limits on natural selection? In P. Moorehead and M. Kaplan, editors, *Mathematical Challenges to the Neo-Darwinian Interpretation of Evolution*, pages 35–46. The Wistar Institute Press, Philadelphia, PA, 1967.

W. Boyer and G. Hura. Non-evolutionary algorithm for scheduling dependent tasks in distributed heterogeneous computing environments. *Journal of Parallel and Distributed Computing*, 65 (9):1035–1046, 2005.

F. Brasilero, A. Duarte, D. Carvalho, R. Barbera, and D. Scardacci. An approach for the coexistence of service and opportunistic grids: The EELA-2 case. In *Proceedings of the 2nd Latin-American Grid Workshop*, Campo Grande, Brazil, 2008.

T. Braun, H. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, and R. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, 2001.

T. Braun, H. Siegel, and A. Maciejewski. Heterogeneous computing: Goals, methods, and open problems. In *Proceedings of the 8th International Conference on High Performance Computing*, pages 307–320, London, UK, 2000. Springer-Verlag.

T. Braun, H. Siegel, A. Maciejewski, and Y. Hong. Static resource allocation for heterogeneous computing environments with tasks having dependencies, priorities, deadlines, and multiple versions. *Journal of Parallel and Distributed Computing*, 68(11):1504–1516, 2008.

Y. Caniou and J. Gay. Simbatch: An API for simulating and predicting the performance of parallel resources managed by batch systems. In *Euro-Par Workshops*, volume 5415 of *Lecture Notes in Computer Science*, pages 223–234. Springer, 2008.

E. Cantú-Paz. *Efficient and accurate parallel genetic algorithms*. Kluwer Academic Publishers, 2000.

J. Carretero and F. Xhafa. Using genetic algorithms for scheduling jobs in large scale grid applications. *Journal of Technological and Economic Development, Vilnius Gediminas Technical University*, 12(1):11–17, 2006.

C. Coello and G. Pulido. A micro-genetic algorithm for multiobjective optimization. In *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*, pages 126–140, London, UK, 2001. Springer-Verlag. ISBN 3540417451.

C. Coello, D. Van Veldhuizen, and G. Lamont. *Evolutionary algorithms for solving multi-objective problems*. Kluwer Academic, New York, 2002. ISBN 0306467623.

P. Crescenzi and V. Kann. A compendium of NP optimization problems. In *Complexity and Approximation*. Springer Verlag, 1999. URL `www.nada.kth.se/~viggo/problemlist/compendium.html`. Consulted July 2009.

A. Croes. A method for solving traveling salesman problems. *Operations Research*, 5:791–812, 1958.

R. Darst. *Introduction to Linear Programming. Applications and Extensions*. CRC Press, New York, 1991.

I. Das and J. Dennis. A closer look at drawbacks of minimizing weighted sums of objectives for Pareto set generation in multicriteria optimization problems. *Structural optimization*, 14(1), 1997.

L. Davis. *Handbook of genetic algorithms*. van Nostrand Reinhold, New York, 1991.

K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley-Interscience Series in Systems and Optimization. John Wiley & Sons, Chichester, 2001.

K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Proceedings of the Parallel Problem Solving from Nature VI Conference*, volume 1917 of *Lecture Notes in Computer Science*, pages 849–858. Springer, 2000.

M. Dhodhi, I. Ahmad, A. Yatama, and I. Ahmad. An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 62(9):1338–1361, 2002.

V. DiMartino and M. Mililotti. Scheduling in a grid computing environment using genetic algorithms. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, page 297, Washington, DC, USA, 2002. IEEE Computer Society.

V. DiMartino and M. Mililotti. Sub optimal scheduling in a grid using genetic algorithms. *Parallel Computing*, 30(5-6):553–565, 2004.

F. Dong and S. Akl. Scheduling algorithms for grid computing: State of the art and open problems. Technical Report 2006-504, School of Computing, Queen's University, Canada, 2006.

B. Duran and F. Xhafa. The effects of two replacement strategies on a genetic algorithm for scheduling jobs on computational grids. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, pages 960–961, New York, NY, USA, 2006. ACM.

J. Durillo, A. Nebro, F. Luna, and E. Alba. A study of master-slave approaches to parallelize NSGA-II. In *22nd IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8, Miami, Florida USA, 2008. IEEE.

J. Durillo, A. Nebro, F. Luna, B. Dorronsoro, and E. Alba. jMetal: A Java Framework for Developing Multi-Objective Optimization Metaheuristics. Technical Report ITI-2006-10, Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, E.T.S.I. Informática, Campus de Teatinos, 2006.

G. Dyson. *Darwin among the Machines: The Evolution of Global Intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997. ISBN 0201406497.

H. El-Rewini, T. Lewis, and H. Ali. *Task scheduling in parallel and distributed systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994. ISBN 0130992356.

M. Eshaghian. *Heterogeneous Computing.* Artech House, Norwood, MA, USA, 1996. ISBN 089065527.

L. Eshelman. The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination. In *Foundations of Genetics Algorithms*, pages 265–283. Morgan Kaufmann Publishers, 1991.

Folding@home. Folding@home statistics page. URL: `http://folding.stanford.edu/English/Stats`, consulted September 2009.

C. Fonseca and P. Fleming. Multiobjective optimization. In T. Bäck, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, chapter C4.5, pages C4.5:1–9. Oxford Univ. Press, 1997.

I. Foster and C. Kesselman. *The Grid: Blueprint for a Future Computing Infrastructure.* Morgan Kaufmann Publishers, 1998. ISBN 1558604758.

R. Fourer, D. Gay, and B. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* Duxbury Press, 2002. ISBN 0534388094.

R. Freund, V. Sunderam, A. Gottlieb, K. Hwang, and S. Sahni. Special issue on heterogeneous processing. *Journal of Parallel and Distributed Computing*, 21(3), 1994.

M. Garey and D. Johnson. *Computers and intractability.* Freeman, 1979. ISBN 0716710447.

F. Glover and G. Kochenberger. *Handbook of Metaheuristics (International Series in Operations Research & Management Science).* Springer, 2003. ISBN 1402072635.

D. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison Wesley, New York, 1989a. ISBN 0201157675.

D. Goldberg. Sizing populations for serial and parallel genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 70–79, San Francisco, CA, USA, 1989b. Morgan Kaufmann.

M. Grajcar. Genetic list scheduling algorithm for scheduling and allocation on a loosely coupled heterogeneous multiprocessor system. In *Proceedings of the 36th ACM/IEEE Conference on Design Automation*, pages 280–285, New York, NY, USA, 1999. ACM.

M. Grajcar. Strengths and weaknesses of genetic list scheduling for heterogeneous systems. In *Proceedings of International Conference on Application of Concurrency to System Design*, pages 123–132. IEEE, 2001.

J. Grefenstette. Parallel adaptive algorithms for function optimization. Technical Report CS-81-19, Vanderbilt University, Computer Science Department, Nashville, TN, 1981.

Grid5000. Grid5000 resource information page. URL: `https://www.grid5000.fr/`, consulted September 2009.

W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, 1996.

W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* MIT Press, Cambridge, MA, 1994. ISBN 0-262-57104-8.

P. Grosso. *Computer simulations of genetic adaptation: Parallel Subcomponent Interaction in a Multilocus Model.* PhD thesis, University of Michigan, USA, 1985.

T. Grüninger and D. Wallace. Multimodal optimization using genetic algorithms: An investigation of a new crowding variation and the relationship between parental similarity and the effect of crossover. Technical report, Department of Mechanical Engineering, MIT, Cambridge, MA, 1997.

F. Hayes-Roth. Review of "Adaptation in Natural and Artificial Systems by John H. Holland", The U. of Michigan Press, 1975. *SIGART Bulletin*, 53:15–15, 1975.

D. Hensgen, T. Kidd, D. St. John, M. Schnaidt, H. Siegel, T. Braun, M. Maheswaran, S. Ali, J. Kim, C. Irvine, T. Levin, R. Freund, M. Kussow, M. Godfrey, A. Duman, P. Carff, S. Kidd, V. Prasanna, P. Bhat, and A. Alhusaini. An overview of MSHN: The management system for heterogeneous networks. In *8th IEEE Workshop on Heterogeneous Computing Systems, San Juan, Puerto Rico, April 1999*, pages 184–198, 1999.

J. Holland. Outline for a logical theory of adaptive systems. *Journal of the ACM*, 9(3):297–314, 1962.

O. Ibarra and C. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM*, 24(2):280–289, 1977. ISSN 0004-5411.

ILOG. *ILOG CPLEX 10.1: User's Manual*. ILOG Inc., Mountain View, California, USA, URL: `http://www.gnu.org/software/glpk/glpk.html`, 2006. Consulted October 2009.

G. Iordache, M. Boboila, F. Pop, C. Stratan, and V. Cristea. A decentralized strategy for genetic scheduling in heterogeneous environments. *Multiagent and Grid Systems*, 3(4):355–367, 2007.

W. Jakob, A. Quinte, K. Stucky, and W. Süß. Optimised scheduling of grid resources using hybrid evolutionary algorithms. In *5th International Conference on Parallel Processing and Applied Mathematics*, pages 406–413, 2005.

A. Khokhar, V. Prasanna, M. Shaaban, and C. Wang. Heterogeneous computing: challenges and opportunities. *Computer*, 26(6):18–27, Jun 1993.

J. Knowles and D. Corne. Approximating the nondominated front using the Pareto Archived Evolution Strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

K. Krishnakumar. Micro genetic algorithms for stationary and nonstationary function optimization. In *Proceedings of the SPIE Intelligent Control and Adaptive Systems Conference*, pages 289–296, Orlando, FL, USA, 1989.

Y. Kwok and I. Ahmad. Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. *Journal of Parallel and Distributed Computing*, 47:58–77, 1997.

Y. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computer Surveys*, 31(4):406–471, 1999.

J. Leung, L. Kelly, and J. Anderson. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, Inc., Boca Raton, FL, USA, 2004. ISBN 1584883979.

H. Li, D. Groep, J. Templon, and L. Wolters. Predicting job start times on clusters. In *Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*, pages 301–308, Washington, DC, USA, 2004. IEEE Computer Society.

S. Lin and B. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operational Research*, 21:498–516, 1973.

P. Luo, K. Lü, and Z. Shi. A revisit of fast greedy heuristics for mapping a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 67(6):695–714, 2007.

A. Makhorin. *GNU Linear Programming Kit, Version 4.9*. GNU Software Foundation, URL: `http://www.gnu.org/software/glpk/glpk.html`, 2006. Consulted October 2009.

M. Matsumoto and T. Nishimura. Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998.

H. Mohamed and D. Epema. KOALA: a co-allocating grid scheduler. *Concurrency and Computation: Practice and Experience*, 20(16):1851–1876, 2008.

G. Neagu, N. Andrei, V. Sima, V. Cristea, C. Nae, D. Petcu, and R. Stanciu. Grid enabled applications for modeling, simulation and optimization. In *Conference CEEX 2007: "Excellence Research as a way to E.R.A"*, pages 1–6, Brasov, Romania, 2007.

A. Nebro, E. Alba, G. Molina, F. Chicano, F. Luna, and J. Durillo. Optimal antenna placement using a new multi-objective chc algorithm. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and Evolutionary Computation*, pages 876–883, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-697-4.

S. Nesmachnow. Evolución en el diseño y clasificación de algoritmos genéticos paralelos. In *Actas de la XXVIII Conferencia Latinoamericana de Informática*, pages 1933–1944, Montevideo, Uruguay, 2002. (Text in Spanish).

S. Nesmachnow. Una version paralela del algoritmo evolutivo para optimizacion multiobjetivo NSGA-II. In *Actas del X Congreso Argentino de Ciencias de Computación*, pages 1933–1944, La Matanza, Argentina, 2004. (Text in Spanish).

S. Nesmachnow. Un algoritmo evolutivo multiobjetivo paralelo aplicado al diseño de redes de comunicaciones confiables. In *Actas del IV Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados*, pages 145–152, Granada, España, 2005. (Text in Spanish).

S. Nesmachnow. Algoritmos evolutivos paralelos para despacho de tareas en entornos heterogéneos. In *Actas del VI Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados*, pages 571–578, Málaga, España, 2009. (Text in Spanish).

S. Nesmachnow, H. Cancela, and E. Alba. Evolutionary algorithms applied to reliable network communication design. *Engineering Optimization*, 39(7):831–855, 2007.

S. Nesmachnow, H. Cancela, and E. Alba. Heterogeneous computing scheduling with evolutionary algorithms. *Soft Computing*, 2010. To appear.

T. Nishimura. Tables of 64-bit mersenne twisters. *ACM Transactions on Modeling and Computer Simulation*, 10(1):348–357, 2000.

M. Nowostawski and R. Poli. Parallel genetic algorithm taxonomy. In *Proceedings of 3rd International Conference on Knowledge-based Intelligent Information Engineering Systems*, pages 88–92, Adelaide, South Australia, 1999. IEEE.

E. Onbaşioglu and L. Özdamar. Optimization of data distribution and processor allocation problem using simulated annealing. *The Journal of Supercomputing*, 25(3):237–253, 2003.

A. Page and T. Naughton. Framework for task scheduling in heterogeneous distributed computing using genetic algorithms. *Artificial Intelligence Review*, 24(3-4):415–429, 2005.

C. Pettey. Diffusion (cellular) models. In T. Bäck, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, chapter C6.4, pages C6.4:1–6. Oxford Univ. Press, 1997.

S. Porto and C. Ribeiro. A tabu search approach to task scheduling on heterogeneous processors under precedence constraints. *International Journal of High Speed Computing*, 7(1):45–71, 1995.

R. Prodan and T. Fahringer. Dynamic scheduling of scientific workflow applications on the grid: a case study. In *Proceedings of the 2005 ACM symposium on applied computing*, pages 687–694, New York, NY, USA, 2005. ACM.

G. Ritchie and J. Levine. A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments. In *Proceedings of the 23rd Workshop of the UK Planning and Scheduling Special Interest Group*, December 2004.

A. Rodríguez and S. Nesmachnow. MOE: un entorno de trabajo para optimizacin multiobjetivo con algoritmos evolutivos. Technical Report RT 09-21, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, 2009. (Text in Spanish).

SETI@home. SETI@home statistics page. URL: `http://setiathome.ssl.berkeley.edu/`, consulted September 2009.

P. Shroff, D. Watson, N. Flann, and R. Freund. Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments. In *Proceedings of the 5th IEEE Heterogeneous Computing Workshop*, pages 98–104, 1996.

B. Singh and S. Bawa. ACO based optimized scheduling algorithm for computational grids. In *Proceedings of the 3rd conference on IASTED International Conference*, pages 283–286, Anaheim, CA, USA, 2007. ACTA Press.

H. Singh and A. Youssef. Mapping and scheduling heterogeneous task graphs using genetic algorithms. In *Proceedings of the 5th IEEE Heterogeneous Computing Workshop*, pages 86–97, 1996.

N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

K. Stucky, W. Jakob, A. Quinte, and W. Süss. Tackling the grid job planning and resource allocation problem using a hybrid evolutionary algorithm. In R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Wasniewski, editors, *7th International Conference on Parallel Processing and Applied Mathematics*, volume 4967 of *Lecture Notes in Computer Science*, pages 589–599. Springer, 2007.

P. Sugavanam, H. Siegel, A. Maciejewski, S. Amjad, M. Al-Otaibi, M. Aydin, K. Guru, A. Horiuchi, Y. G. Krishnamurthy, P. Lee, A. Mehta, M. Oltikar, R. Pichel, A. Pippin, M. Raskey, V. Shestak, and J. Zhang. Processor allocation for tasks that is robust against errors in computation time estimates. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, pages 122–135, Washington, DC, USA, 2005. IEEE Computer Society.

P. Sugavanam, H. Siegel, A. Maciejewski, M. Oltikar, A. Mehta, R. Pichel, A. Horiuchi, V. Shestak, M. Al-Otaibi, Y. Krishnamurthy, S. Ali, J. Zhang, M. Aydin, P. Lee, K. Guru, M. Raskey, and A. Pippin. Robust static allocation of resources for independent tasks under makespan and dollar cost constraints. *Journal of Parallel and Distributed Computing*, 67(4):400–416, 2007.

TeraGrid. TeraGrid resource information page. URL: `http://www.teragrid.org/userinfo/hardware/resources.php`, consulted September 2009.

M. Theys, T. Braun, H. Siegel, A. Maciejewski, and Y. Kwok. Mapping tasks onto distributed heterogeneous computing systems using a genetic algorithm approach. In *Solutions to parallel and distributed computing problems*, pages 135–178, New York, USA, 2001. Wiley.

Y. Tirat-Gefen and A. Parker. MEGA: An approach to system-level design of application-specific heterogeneous multiprocessors. In *Proceedings of the 5th IEEE Heterogeneous Computing Workshop*, pages 105–117, 1996.

H. Tupcouglu, S. Hariri S., and M. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3): 260–274, 2002.

L. Wang, H. Siegel, V. Roychowdhury, and A. Maciejewski. Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *Journal of Parallel and Distributed Computing*, 47(1):8–22, November 1997.

D. Whitley and A. Sutton. Partial neighborhoods of elementary landscapes. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 381–388, New York, NY, USA, 2009. ACM.

D. Whitley, A. Sutton, and A. Howe. Understanding elementary landscapes. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 585–592, New York, NY, USA, 2008. ACM.

WLCG. WLCG resource information page. URL: `http://lcg.web.cern.ch/LCG/resources.htm`, consulted September 2009.

A. Wu, H. Yu, S. Jin, K. Lin, and G. Schiavone. An incremental genetic algorithm approach to multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 15(9): 824–834, 2004.

F. Xhafa. *A Hybrid Evolutionary Heuristic for Job Scheduling in Computational Grids*, chapter 10. Springer Verlag Series: Studies in Computational Intelligence, Vol. 75, 2007. ISBN 3-540-73296-9.

F. Xhafa, E. Alba, B. Dorronsoro, and B. Duran. Efficient batch job scheduling in grids using cellular memetic algorithms. *Journal of Mathematical Modelling and Algorithms*, 7(2):217–236, 2008a.

F. Xhafa, L. Barolli, and A. Durresi. An experimental study on GA replacement operators for scheduling on grids. *Journal of Interconnection Networks*, 8(4):427–443, 2007a.

F. Xhafa, J. Carretero, and A. Abraham. Genetic algorithm based schedulers for grid computing systems. *International Journal of Innovative Computing Information and Control*, 3(5):1–19, 2007b.

F. Xhafa, J. Carretero, E. Alba, and B. Dorronsoro. Design and evaluation of tabu search method for job scheduling in distributed environments. In *Proceedings of the 21th International Parallel and Distributed Processing Symposium*, pages 1–8. IEEE Computer Society, 2008b.

F. Xhafa and B. Duran. Parallel memetic algorithms for independent job scheduling in computational grids. In C. Cotta and J. van Hemert, editors, *Recent Advances in Evolutionary Computation for Combinatorial Optimization*, volume 153 of *Studies in Computational Intelligence*, pages 219–239. Springer, 2008.

F. Xhafa, B. Duran, A. Abraham, and K. Dahal. Tuning struggle strategy in genetic algorithms for scheduling in computational grids. In *Proceedings of the 7th Computer Information Systems and Industrial Management Applications*, pages 275–280, Washington, DC, USA, 2008c. IEEE Computer Society.

J. Yang, I. Ahmad, and A. Ghafoor. Estimation of execution times on heterogeneous supercomputer architectures. In *Proceedings of the 1993 International Conference on Parallel Processing*, pages 219–226, Washington, DC, USA, 1993. IEEE Computer Society.

A. YarKhan and J. Dongarra. Experiments with scheduling using simulated annealing in a grid environment. In *Proceedings of the 3rd International Workshop on Grid Computing*, pages 232–242, London, UK, 2002. Springer-Verlag.

J. Yu and R. Buyya. A budget constrained scheduling of workflow applications on utility grids using genetic algorithms. In *Workshop on Workflows in Support of Large-Scale Science, Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing*. IEEE CS Press, 2006.

E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology, Zurich, Switzerland, 2001.

E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the Strength Pareto Evolutionary Algorithm. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.

A. Zomaya and Y. Teh. Observations on using genetic algorithms for dynamic load-balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(9):899–911, 2001.