PEDECIBA INFORMÁTICA

Instituto de Computación, Facultad de Ingeniería

Universidad de la República

Montevideo, Uruguay

TESIS DE MAESTRÍA EN INFORMÁTICA

Energy-aware scheduling in heterogeneous computing systems

Santiago Iturriaga

siturria@fing.edu.uy

Abril de 2013

Director Académico:

Sergio Nesmachnow, Universidad de la República.

Directores de Tesis:

Sergio Nesmachnow, Universidad de la República.
Bernabé Dorronsoro, Universidad de Lille.

Tribunal:

Dr. Franco Robledo, Universidad de la República, presidente.
Dr. Esteban Mocskos, Universidad de Buenos Aires.
Dr. Francisco Luna, Universidad Carlos III de Madrid, revisor.

Energy-aware scheduling in heterogeneous computing systems
Iturriaga, Santiago
ISSN 0797-6410
Tesis de Maestría en Informática
PEDECIBA
Instituto de Computación - Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay, abril de 2013

Energy aware scheduling in heterogeneous computing systems

Abstract

In the last decade, the grid computing systems emerged as useful provider of the computing power required for solving complex problems. The classic formulation of the scheduling problem in heterogeneous computing systems is NP-hard, thus approximation techniques are required for solving real-world scenarios of this problem. This thesis tackles the problem of scheduling tasks in a heterogeneous computing environment in reduced execution times, considering the schedule length and the total energy consumption as the optimization objectives. An efficient multithreading local search algorithm for solving the multi-objective scheduling problem in heterogeneous computing systems, named ME-MLS, is presented. The proposed method follows a fully multi-objective approach, applying a Pareto-based dominance search that is executed in parallel by using several threads. The experimental analysis demonstrates that the new multithreading algorithm outperforms a set of fast and accurate two-phase deterministic heuristics based on the traditional MinMin. The new ME-MLS method is able to achieve significant improvements in both makespan and energy consumption objectives in reduced execution times for a large set of testbed instances, while exhibiting very good scalability. The ME-MLS was evaluated solving instances comprised of up to 2048 tasks and 64 machines. In order to scale the dimension of the problem instances even further and tackle large-sized problem instances, the *Graphical Processing Unit* (GPU) architecture is considered. This line of future work has been initially tackled with the gPALS: a hybrid CPU/GPU local search algorithm for efficiently tackling a single-objective heterogeneous computing scheduling problem. The gPALS shows very promising results, being able to tackle instances of up to 32768 tasks and 1024 machines in reasonable execution times.

Keywords: Metaheuristic algorithms, Scheduling, Heterogeneous computing, Grid computing

Planificación en sistemas heterogéneos considerando eficiencia energética

Resumen

En la última década, los sistemas de computación grid se han convertido en útiles proveedores de la capacidad de cálculo necesaria para la resolución de problemas complejos. En su formulación clásica, el problema de la planificación de tareas en sistemas heterogéneos es un problema NP-difícil, por lo que se requieren técnicas de resolución aproximadas para atacar instancias de tamaño realista de este problema. Esta tesis aborda el problema de la planificación de tareas en sistemas heterogéneos, considerando el largo de la planificación y el consumo energético como objetivos a optimizar. Para la resolución de este problema se propone un algoritmo de búsqueda local eficiente y multihilo. El método propuesto se trata de un enfoque plenamente multiobjetivo que consiste en la aplicación de una búsqueda basada en dominancia de Pareto que se ejecuta en paralelo mediante el uso de varios hilos de ejecución. El análisis experimental demuestra que el algoritmo multithilado propuesto supera a un conjunto de heurísticas deterministas rápidas y eficaces basadas en el algorítmo MinMin tradicional. El nuevo método, ME-MLS, es capaz de lograr mejoras significativas tanto en el largo de la planificación y como en consumo energético, en tiempos de ejecución reducidos para un gran número de casos de prueba, mientras que exhibe una escalabilidad muy promisoria. El ME-MLS fue evaluado abordando instancias de hasta 2048 tareas y 64 máquinas. Con el fin de aumentar la dimensión de las instancias abordadas y hacer frente a instancias de gran tamaño, se consideró la utilización de la arquitectura provista por las unidades de procesamiento gráfico (GPU). Esta línea de trabajo futuro ha sido abordada inicialmente con el algoritmo gPALS: un algoritmo híbrido CPU/GPU de búsqueda local para la planificación de tareas en en sistemas heterogéneos considerando el largo de la planificación como único objetivo. La evaluación del algoritmo gPALS ha mostrado resultados muy prometedores, siendo capaz de abordar instancias de hasta 32768 tareas y 1024 máquinas en tiempos de ejecución razonables.

Palabras clave: Algoritmos metaheurísticos, Planificación de tareas, Computación heterogénea, Computación grid

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Heterogeneous computing (HC) systems usually comprise a large number of heterogeneous computing resources which are able to work cooperatively. In the last decade, heterogeneous computing systems have emerged as useful providers of the computing power needed to solve complex problems arising in many areas of application. Since their emergence, heterogeneous computing systems have become larger and larger mainly because of the ever demanding scientific community and the information technology industry, and thanks to the fast increase of computing power and the rapid development of high-speed networking. Despite continued technological improvements, computing resources fail to keep up with demand, specially with the scientific community demand. No matter the state-of-the-art of the computing technology, the scientific community always demands more computing power in order to perform bigger simulations, obtain more precise results, or model wider realities. In this regard, heterogeneous computing systems provide a mean to tackle problems of increased complexity by enabling a large-scale collaboration of distributed computing resources (Foster and Kesselman, 2003).

In current distributed HC systems, efficiently scheduling tasks to be executed on the available resources of the computing infrastructure is a key problem in order to make the most of the available computing power. The goal of the scheduling problem is to assign tasks to computing resources satisfying some specific efficiency criteria, usually related to the total execution time of a bunch of tasks, but frequently also considering other relevant objectives such as resource utilization, energy consumption, quality of service, etc.

Traditional scheduling problems are NP-hard (Garey and Johnson, 1979), thus classic exact methods are not useful in practice to solve large instances of such problems. Many deterministic scheduling heuristics have been proposed, but usually they do not scale appropriately when solving large dimension instances. This motivated the use of non-deterministic heuristics and metaheuristics in order to tackle scheduling problems. These non-deterministic approaches have proven to be able to compute efficient schedules in reasonable execution time for a wide range of dimension instances (Nesmachnow et al., 2010; Pinel et al., 2013; Xhafa et al., 2008b).

In recent years, energy consumption has become a major concern in large data centers. Electricity costs have increased so much that engineers from Google's data centers warned that if power consumption continues to grow, electricity costs would overtake hardware costs by a large margin (Barroso, 2005).

Google is not the only one concerned by electricity costs, according to Amazon (Hamilton, 2009) the energy-related costs represent approximately 42% of the total budget in a computing center when considering a 15 years amortization; 19% of the energy-related budget is dedicated to direct computing power consumption, and the remaining 23% of the energy-related budget is dedicated to indirect cooling-related infrastructure. Processors are the main consumers of energy in such systems, but they also offer the most flexible energy management mechanisms, by applying dynamic voltage scaling (DVS), dynamic power management, slack sharing and reclamation, etc. (Khan and Ahmad, 2009; Kim et al., 2007; Zhu et al., 2003). Reducing processors consumption is a great challenge, and many researchers currently focus on the development of energy-aware scheduling algorithms for HC systems (Lee and Zomaya, 2009).

This thesis tackles the problem of scheduling tasks in HC environments considering the schedule length and the total energy consumption as the optimization objectives. A highly efficient multithreading population-based local search method (ME-MLS) is proposed to find a set of accurate trade-off solutions for the proposed multi-objective scheduling problem. ME-MLS is a population-based method that follows a fully multi-objective approach; it does not optimize an aggregated function of the problem objectives, but uses a Pareto-based dominance analysis instead. The two most critical components of ME-MLS are its embedded local search procedure, and its archiving algorithm. The local search procedure is responsible for improving the population of solutions maintained during the algorithm execution. On the other hand, the archiving algorithm is responsible for choosing which of all the solutions computed by the local search are worth maintaining during the algorithm execution. The ME-rPALS is the embedded local search procedure proposed for the ME-MLS. The ME-rPALS was designed to be an efficient non-deterministic local search heuristic for tackling combinatorial optimization problems. As for the ME-MLS archiving algorithm, two different algorithms are proposed to be used. The efficacy, efficiency, and scalability of the ME-MLS using each of the proposed archiving algorithms were evaluated. Both variants are compared with well-known deterministic heuristics over a large set of instances. The experimental results show that ME-MLS is able to compute an adequate set of accurate trade-off schedules in short execution times.

The ME-MLS was evaluated solving instances comprised of up to 2048 tasks and 64 machines. One line of future work aims at using the GPU architecture for scaling the dimension of the problem instances even further. The GPU architecture for general purpose computing is an emerging technology which is able to provide the computing power required for tackling very large problem instances. Taking advantage of the GPU architecture, the gPALS algorithm was initially proposed. The gPALS algorithm is an hybrid CPU/GPU local search method for tackling the problem of scheduling tasks in HC environments considering only the schedule length as the optimization objective. The efficacy, efficiency, and scalability of the proposed gPALS are also evaluated, but tackling a much smaller set of HCSP instances. The gPALS is compared with well-known deterministic heuristics and with the cellular EA proposed by Pinel et al. (2013). The experimental results show that gPALS is able to compute accurate schedules and provide very significant acceleration rates. These results present the GPU architecture as a very promising architecture for tackling large-sized instances of the energy-aware scheduling problem in heterogeneous computing systems.

The content of this thesis is structured as follows. The next chapter presents a brief introduction to scheduling problems theory. It introduces heterogeneous computing systems and discusses why energy consumption is such a critical issue. It presents the heterogeneous computing scheduling problem in energy-aware environments, it presents some computing model for the problem, and a set of techniques for tackling it. Chapter 3 introduces metaheuristic and non-deterministic algorithms for solving NP-hard optimization problems. The fundamentals of multi-objective optimization are introduced, along with a set of metrics for measuring multi-objective optimization results. Finally, two local search methods are presented, one for solving single-objective optimization problems, and the other for solving multi-objective optimization problems. Chapter 4 reviews and comments previous works that have recently tackled energy-aware scheduling problems. The reviewed works are organized according to whether they tackle a single-objective optimization problem, a multi-objective optimization problem using a single-objective approach, or a true multi-objective optimization problem. Chapter 5 describes the ME-MLS algorithm design details, the solution encoding, the archiving algorithms, and the embedded local search design. Chapter 6 presents the experimental analysis and the discussion of the reported results for the ME-MLS algorithm. It presents the execution platform, the problem instances, the methods for computing a baseline reference, the parameter setting experiments results, the numerical results of the ME-MLS algorithm and the comparison with the baseline reference methods. Chapter 7 addresses the problem of scheduling very large problem scenarios. It introduces the GPU computing architecture, and presents the gPALS, a hybrid CPU/GPU local search algorithm. An experimental analysis is performed on gPALS for evaluating its numerical efficiency and its parallel performance. Finally, Chapter 8 presents the conclusions and future work.

Publications issued from this thesis work

The next list presents a brief summary of the publications issued from the research performed during this thesis.

- S. Iturriaga, S. Nesmachnow, and B. Dorronsoro. A Multithreading Local Search For Multiobjective Energy-Aware Scheduling In Heterogeneous Computing Systems. In *Proceedings of the 26th European Conference on Modelling and Simulation (ECMS)*, pages 497–503, Koblenz, Germany, 2012a. ISBN 978-0-9564944-4-3

  This article introduces an efficient multithreading local search algorithm for solving the multiobjective scheduling problem in heterogeneous computing systems considering the makespan and energy consumption objectives. The proposed method follows a fully multiobjective approach using a Pareto-based dominance search executed in parallel. The experimental analysis demonstrates that the new multithreading algorithm outperforms a set of deterministic heuristics based on Min-Min. The new method is able to achieve significant improvements in both objectives in reduced execution times for a broad set of testbed instances.

- S. Iturriaga, S. Nesmachnow, F. Luna, and E. Alba. A parallel online GPU scheduler for large heterogeneous computing systems. In *Proceedings of the 5th High-Performance Computing Latin America Symposium (HPCLatAm)*, JAIIO '12, Buenos Aires, Argentina, 2012b

  This work presents a parallel implementation on GPU for a stochastic local search method to efficiently solve the task scheduling problem in heterogeneous computing environments. The research community has been searching for accurate schedulers for heterogeneous computing systems, able to run in reduced times. The parallel stochastic search proposed in this work is based on simple operators in order to keep the computational complexity as low as possible, thus allowing large scheduling instances to be efficiently tackled. The experimental analysis demonstrates that the parallel stochastic local search method on GPU is able to compute accurate suboptimal schedules in significantly shorter execution times than state-of-the-art schedulers.

- S. Iturriaga, S. Nesmachnow, and C. Tutté. Metaheuristics for multiobjective energy-aware scheduling in heterogeneous computing systems. In *EU/Metaheuristics Meeting Workshop (EU/ME)*, Copenhaguen, Denmark, 2012c

  This article reports the advances on applying metaheuristic algorithms to solve the scheduling problem that proposes the simultaneus optimization of makespan and energy consumption in HC systems. The proposed methods include Multithreading Local Search (MLS), a highly efficient multiobjective local search; and two well-known multiobjective evolutionary algorithms (MOEAs), namely NSGA-II and SPEA2. The three methods follow a fully multiobjective approach, since they do not optimize an aggregated function of the problem objectives, but they use Pareto-based dominance techniques in the optimization.

- S. Iturriaga, S. Nesmachnow, B. Dorronsoro, and P. Bouvry. Energy efficient scheduling in heterogeneous systems with a parallel multiobjective local search. *Computing and Informatics Journal (CAI)*, 2013a. Accepted on November 2012, to appear

  This article introduces ME-MLS, an efficient multithreading local search algorithm for solving the multiobjective scheduling problem in heterogeneous computing systems. We consider the minimization of both the makespan and energy consumption objectives. The proposed method follows a fully multiobjective approach, applying a Pareto-based dominance search that is executed in parallel by using several threads. The experimental analysis demonstrates that the new multithreading algorithm outperforms a set of fast and accurate two-phases deterministic heuristics based on the traditional MinMin. The new ME-MLS method is able to achieve significant improvements in both makespan and energy consumption objectives in reduced execution times for a large set of testbed instances, while exhibiting a near linear speedup behavior when using up to 24 threads.

- S. Iturriaga, S. Nesmachnow, F. Luna, and E. Alba. A parallel local search in CPU/GPU for scheduling independent tasks on large heterogeneous computing systems. *Journal of Parallel and Distributed Computing (JPDC)*, 2013b. Submitted on January 2013, pending acceptance

  This article presents the parallel implementation on CPU/GPU of two variants of a stochastic local search method to efficiently solve the problem of scheduling independent tasks in heterogeneous computing environments. The research community has been searching for accurate schedulers for heterogeneous computing systems, able to execute in reduced times. The parallel stochastic search on CPU/GPU proposed in this article is based on a set of simple operators in order to keep the computational complexity as low as possible, thus allowing large instances of the scheduling problem to be efficiently tackled. The experimental analysis demonstrates that both versions of the parallel stochastic local search method on CPU/GPU are able to compute accurate suboptimal schedules in significantly shorter execution times than state-of-the-art schedulers. The proposed methods also outperform a parallel evolutionary scheduler recently published in the literature, in terms of both efficiency and solution quality.

- S. Iturriaga, P. Ruiz, S. Nesmachnow, B. Dorronsoro, and P. Bouvry. A Parallel Multi-objective Local Search for AEDB Protocol Tuning. In *Proceedings of the 16th International Workshop on Nature Inspired Distributed Computing*, in the 27th IEEE/ACM International Parallel & Distributed Processing Symposium, Boston, Massachusetts, USA, 2013c. Accepted on February 2013, to appear

  This work presents a stochastic local search method for efficiently solve the scheduling problem in heterogeneous computing environments. The research community has been searching for accurate schedulers for heterogeneous computing systems, able to perform in reduced times. The stochastic search proposed in this work is based on simple operators in order to keep the computational complexity as low as possible, thus allowing to efficiently tackle large scheduling instances. The experimental analysis demonstrates that the new stochastic local search method is able to compute accurate suboptimal schedules in significantly shorter execution times than state-of-the-art schedulers.

Chapter 2

Heterogeneous computing scheduling problem

This chapter introduces the reader to scheduling problems in general, and presents both the single-objective *heterogeneous computing scheduling problem* (HCSP) and the bi-objective *makespan-energy heterogeneous computing scheduling problem* (ME-HCSP). Next, the scheduling problem formulation, the models for the HC system, and the problem instances for both HCSP versions are presented. In the final sections, different exact and approximation methods for solving the HCSP are reviewed.

2.1 Scheduling problems

Scheduling problems are combinatorial optimization problems which consist in allocating a limited amount of resources to some activities over a period of time. Activities may be tasks in computing environments, flight routes in an airline, patients in a hospital, etc. Resources may be machines to execute the tasks, airplanes to fly the routes, doctors to attend the patients, etc. (**?**).

The *resource-constrained project scheduling problem* (RCPSP) (Garey and Johnson, 1979) is a very general scheduling problem that may be used to model many other scheduling problems. In the RCPSP formulation there are n activities $i = 1, ..., n$ and m renewable resources $j = 1, ..., m$. Each resource j has a constant amount of R_j units of resource available. Activity i requires p_i units of time to be processed, and requires r_{ij} units of resources from its assigned resource j. A resource j is called disjunctive if $R_j = 1$, otherwise it is called cumulative. If resource j is disjunctive, only one activity at a time can be processed in j. Precedence constraints are defined between activities such that if v precedes u it is noted $v \rightarrow u$, then task v must be completed before u. The problem objective is to determine the resource allocation and the starting time (S_i) for every activity $i = 1, ..., n$ in order to minimize the maximum completion time, constrained by the available resource units and by the defined task precedences. The maximum completion time (or *makespan*) is defined as the time spent since the first task begins execution to the moment when the last task is completed. In the aforementioned problem the makespan can be mathematically formulated as $C_{max} = \max_{i=1}^{n} \{S_i + p_i\}$.

According to Leung et al. (2004), there are many different properties which can be used to characterize a scheduling problem. For instance, an algorithm may or may not have complete knowledge of all the activities to be scheduled beforehand.

When the algorithm has complete knowledge of the activities to be scheduled, the problem is classified as an *offline* scheduling problem. On the other hand, in an *online* scheduling problem, activities dynamically appear over time, and known activities must be scheduled without knowledge of any possible future activities. Furthermore, in an *online* scheduling problem one may not know the complete characteristics of the activities before processing one of them, e.g. the processing time of each activity may be unknown to the scheduler. When the complete job characteristics are available as inputs to the scheduling algorithm, the problem is said to be *clairvoyant*, otherwise the problem is called *non-clairvoyant*. Clairvoyant scheduling problems usually arise in environments where activities cannot be preempted, that is, environments where activities cannot be interrupted after their execution has begun. Batch scheduling problems, such as manufacturing scheduling, are classic clairvoyant scheduling problems. On the other hand, non-clairvoyant scheduling problems are usually present in preemptive scheduling problems, where activities can be interrupted and displaced in favor of others activities. Non-clairvoyant scheduling problems arise in environments like task scheduling in operating systems.

In order to standardize the notation, Graham et al. (1979) introduced a characterization scheme for scheduling problems which follows the $\alpha|\beta|\gamma$ form. In this notation, the α field details the machine infrastructure, the β field specifies the job characteristics, and the γ entry represents the optimization criterion. For example, a problem in which the field α has a value of 1 represents a single machine infrastructure, when α equals P denotes m parallel identical machines, α equals Q denotes m parallel uniform machines, etc. The second field, the β field, specifies job characteristics such as precedences, release times, preemption, etc. Finally, the γ field stands for the objective function, e.g.: C_{max} for makespan, which measures the maximum completion time; L_{max} for maximum lateness, which measures the worst violation of the due dates; $\sum w_j C_j$ for total weighted completion time, etc. (**?**)

Methods for solving scheduling problems depend on the computational complexity of the particular problem. Most scheduling problems are NP-hard (Knust and Brucker, 2006), and as such, no known algorithm is capable of solving them in polynomial time. Because of this, exact methods are not useful in realistic scheduling problems, requiring the use of approximate solutions algorithms in order to solve this kind of problems in reasonable time. Usual approximate solutions algorithms include polynomial time approximation schemes (Leung et al., 2004) and non-deterministic heuristic approaches (Pinedo, 2008). In this context, an algorithm is said to be c-competitive if in the worst case the computed objective function value is at most a factor c away from the optimal schedule.

2.2 Heterogeneous computing systems

A heterogeneous computing (HC) system can be seen as a virtual computer comprised of a set of distributed heterogeneous machines that contribute their individual computational power to the computational power of the aggregated system. An HC system can be characterized by the following features (Kshemkalyani and Singhal, 2008): *i) no common physical clock* is globally available in the system, showing the inherent asynchrony amongst the processors; *ii) no shared memory* is available in the system requiring a message-passing communication model; *iii)* the comprised machines are *autonomous and heterogeneous* allowing them be loosely coupled in the way that having different speeds, running different operating systems, etc.; *iv)* the comprised machines are *geographically separated*, although it is not necessary for the machines to be connected via a wide-area network.

Heterogeneous computing systems range from small *clusters of workstations* (COW) networked using a local-area network (LAN), to huge *computing grids* comprising thousands of processors distributed on many computing centers interconnected over a wide-area network (WAN). In many application areas, heterogeneous computing systems have become useful providers of the computing power needed for the execution of scientific and high performance applications (Foster and Kesselman, 2003; Zhao et al., 2008).

Many difficulties arise when using a HC system, a crucial problem to tackle in HC systems consists in scheduling the execution of the user submitted tasks in order to efficiently use the system resources. The efficient utilization of system resources can be defined in many different ways and can address many different objectives, it can account for the tasks execution times, for the quality of service of the system, for the economic profit, for the energy consumption, etc. (Buyya, 2002; Nesmachnow et al., 2010).

2.2.1 Energy-aware heterogeneous computing systems

In the last decade, the energy consumption in many computational systems has increased considerably and has become an expensive resource, specially in large heterogeneous computing systems (Fan et al., 2007). This increase in energy consumption is strongly related with Moore's law which states that the number of transistors that can be placed on an integrated circuit doubles approximately every 18 months; transistors consume energy, so more transistors imply more energy consumption. Furthermore, consumed energy is transformed into heat, requiring bigger energy-consuming cooling systems to maintain the computing systems at an operating temperature (**?**).

To deal with the aforementioned energy consumption problem, two different approaches to explicitly control the consumed energy have appeared in the literature. The first one consists in powering down some machines of the HC system when the system is lightly loaded (Orgerie et al., 2008). The second approach makes use of the Dynamic Voltage Scaling (DVS), a technique originally proposed by Burd et al. (2000). DVS is included on most modern microprocessors and allows them to run at variable speeds, reducing their energy consumption when running at slower speeds. Including DVS control in scheduling algorithms researchers have been able to lower the energy consumption of the infrastructure while meeting the usability requirements (Kim et al., 2007; Lee and Zomaya, 2009; von Laszewski et al., 2009). Both of the aforementioned energy-saving techniques degrade the computing performance of a HC system.

2.3 Energy-aware heterogeneous computing scheduling problem

The more general *heterogeneous computing scheduling problem* (HCSP) arises when given a set of independent tasks, we want to execute them efficiently in a heterogeneous computing system (Nesmachnow et al., 2010). In the case of the *makespan-energy heterogeneous computing scheduling problem* (ME-HCSP), the efficiency of the schedule is defined in terms of the execution time of the tasks and the energy consumption of the system. The ME-HCSP is generalization of the RCPSP (previously stated in Section 2.1).

2.3.1 Problem formulation

The ME-HCSP follows a non-preemptive offline model which assumes all tasks to be known to the algorithm at scheduling time and assumes every task to be atomic, meaning that a task cannot be interrupted once it began its execution. In the ME-HCSP all tasks are independent, so there is no precedence requirements between tasks. The machines are not identical, meaning that the execution time and the energy consumption required to execute each task varies from one machine to another. Finally, the ME-HCSP follows an unrelated machine model in which each machine can process each task at a different speed. The unrelated machine model defines no relationship between the execution time of a task and its executing machine, that is, a given machine m_j may be faster than another machine m_k when executing some tasks, but m_j may be slower than m_k for other tasks (i.e. the execution times are *unrelated* with respect to the executing machine) (Nesmachnow et al., 2012b).

The formulation of the classic HCSP targets the minimization of the makespan metric aiming to reduce the time required to execute a set of tasks (Nesmachnow et al., 2010).In this work, we tackle the *makespan-energy heterogeneous computing scheduling problem* (ME-HCSP), a variation of the classic HCSP in which the minimization of the makespan metric and the minimization total energy consumption of the system are both optimized.

The mathematical model of ME-HCSP considers the following elements:

- A heterogeneous computing system is composed of a set of heterogeneous machines $P = \{m_1, \ldots, m_M\}$, each machine performing at a certain processing speed and energy consumption.
- A collection of tasks $T = \{t_1, \ldots, t_N\}$ to be executed on the system.
- An *execution time function* $ET : T \times P \to \mathbf{R}^+$, where $ET(t_i, m_j)$ is the time required to execute task t_i on machine m_j.
- An *energy consumption function* $EC : T \times P \to \mathbf{R}^+$, where $EC(t_i, m_j)$ is the energy required to execute task t_i on machine m_j.
- An *idle energy consumption function* $EC_{idle} : P \to \mathbf{R}^+$, being $EC_{idle}(m_j)$ the energy that machine m_j consumes per time unit when it is in idle state.
- A scheduling function $f : T \to P$, which states that task t_i is to be executed by machine m_j only if $f(t_i) = m_j$.

The ME-HCSP aims at finding the scheduling function f that simultaneously minimizes the *makespan* (C_{max}) and the *total energy consumption* (E). The *total energy consumption* considers the energy consumed by the machines, both when executing tasks and when in idle state. The makespan objective is defined in Equation 2.1 and the total energy consumption is defined in Equation 2.2.

$$C_{max} = \max_{m_j \in P} C_j \ , \ with\ C_j = \sum_{\substack{t_i \in T: \\ f(t_i) = m_j}} ET(t_i, m_j) \tag{2.1}$$

$$E = \sum_{t_i \in T} EC(t_i, f(t_i)) + \left\{ \sum_{m_j \in P} (C_{max} - C_j) \times EC_{idle}(m_j) \right\} \tag{2.2}$$

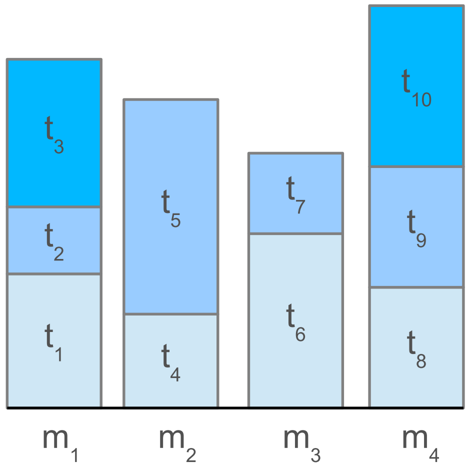


Figure 2.1: Scheduling function example of 10 tasks assigned to 4 machines.

Although the formulation of the problem does not explicitly defines it, a very important characteristic of every scheduler algorithm which aims to tackle the HCSP or the ME-HCSP, is its execution time. Any of these algorithms must be able to execute in reduced time, because the goal of both problems is to minimize the execution time of the tasks in the system, and the execution time of the scheduling algorithm is a time overhead in the system. According to the reviewed works in the literature, an accepted reasonable time for the execution of the scheduling algorithm is in the order of 90 seconds or less (Kołodziej et al., 2011; Nesmachnow et al., 2012b; Xhafa, 2007; Xhafa et al., 2008a). This is usually regarded as an adequate time span for scheduling tasks in realistic HC systems where tasks with execution times in the order of minutes, hours, and even days, are submitted for execution.

Figure 2.1 is a Gantt-like chart example representing a scheduling function of 10 tasks assigned to 4 machines. Note that the execution order of the tasks assigned to a given machine is not relevant, since neither the makespan nor total energy consumption objective functions are affected by the task ordering.

Both the makespan and the total energy consumption depend on the time required to execute the task t_i in its assigned machine m_j. Despite this, both energy consumption and makespan objectives are in conflict in heterogeneous computing systems. To show this, suppose a subset of machines $P' \subset P$ consume less energy per time unit than the rest of the machine $(P \setminus P')$. In this scenario, if we aim to minimize the energy consumption, clearly the best schedule would be to assign all the collection of tasks T to be executed by the machines P'. But in this schedule the makespan objective would suffer because only a subset of the whole computing power of the system would used for task processing. On the other hand, if we use the whole set of machines, the total energy consumption will suffer because of the utilization of the machines that consume the most energy. Hence, there is a conflict between both objectives.

In the aforementioned ME-HCSP formulation, no dependency constraints are defined between tasks so all the tasks can be executed disregarding the execution order. This kind of programs, known as *independent* tasks or *bag of tasks*, are frequent in e-Science applications over heterogeneous computing systems. Some examples of this kind of applications are Single-Program Multiple-Data (SPMD) applications used for multimedia processing, data mining, parallel domain decomposition of numerical models for physical phenomena, etc. Furthermore, in the ME-HCSP formulation every task is sequential, meaning each task requires only one computational resource to be successfully executed.

This scenario might seem too restrictive, nevertheless, the are several studies which show that sequential independent tasks are widely used in grid computing systems (Christodoulopoulos et al., 2008; Iosup and Epema, 2011; Iosup et al., 2006), thus the ME-HCSP faced in this work is relevant in realistic distributed heterogeneous computing systems.

Just like the HCSP, the ME-HCSP follows an unrelated machine model which helps to model the heterogeneity of the system. For example, the unrelated machine model can model scenarios where there are some machines with slow input/output (I/O) access but with fast computing (CPU) speed, and there are other machines with fast I/O access and slow CPU speed. In these scenarios the first class of machines will execute I/O-bound tasks much slower than the second class of machines, but they will execute CPU-bound tasks much faster than the second class of machines. This machine dependent task execution time can only be modeled by using an unrelated machine model.

Using the 3-field notation from Graham et al. (1979), the ME-HCSP is denoted $R||(C_{max}, E)$ representing a scheduling problem with m parallel unrelated machines, an independent collection of tasks with no special characteristics, and two simultaneous objective function (makespan and total energy consumption).

It was shown that $R||C_{max}$ is NP-hard in the strong sense (Leung et al., 2004). Since $R||C_{max}$ is clearly reducible to $R||(C_{max}, E)$; then $R||(C_{max}, E)$ must be NP-hard in the strong sense too.

2.3.2 Models for heterogeneous computing systems

Two different models were considered in this work, one for modeling HCSP scenarios and one for modeling the ME-HCSP scenarios. The first one is the well known *expected time to compute* (ETC) model for the HCSP introduced by Ali et al. (2000), and the second is the *energy consumption in multi-core computers* (EMC) model for the ME-HCSP proposed by Nesmachnow et al. (2012a).

The ETC model has been widely used by the research community when facing task scheduling problems (Ucar et al., 2006; Xhafa and Abraham, 2010). The model defines an unrelated estimation model for the execution time of a collection of tasks in a HC system. It is assumed that an estimation of the computational requirements of each task exists, and the computing speed of each resource in the HC system is known. This data is stored in an ETC matrix of size number of tasks by number of machines ($T \times P$). Each position ETC_{ij} in the matrix details the expected time to compute the task t_i in the machine m_j.

Each problem scenario in the ETC model is classified according to its dimension, machine heterogeneity, task heterogeneity, and consistency. The model *dimension* defines the number of the tasks to be scheduled and the number of machines available in the system, the dimension is specified as num. of tasks $\times$ num. of machines (e.g. 512×16, 1024×32, etc.). *Machine heterogeneity* evaluates the variation of execution times for a given task across the HC resources. A system with similar computing resources has low machine heterogeneity, while high machine heterogeneity represents HC systems with computing resources of very different computing power. *Task heterogeneity* represents the variation of the tasks execution times for a given machine.

In a high task heterogeneity scenario, different types of applications are submitted to execution, from simple programs to complex tasks which require large CPU times to be performed. On the other hand, low task heterogeneity models those scenarios where the tasks computational requirements, and thus their execution times, are similar for a given machine. *Consistency* represents the fact that some machines with special characteristics may be more adequate to execute some jobs than others. 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 corresponds to an ideal case where the execution time of each task is mainly determined by the computational power of each machine, and no other machine characteristics (such as local storage access times) or external factors (such as networking connectivity speed) may affect the task execution time. An *inconsistent* ETC scenario lacks of structure among the computing demands of tasks and the computing power of machines.In this scenario, a machine m_j may be faster than another machine m_k when executing some tasks, but m_j may be slower than m_k for other tasks. This category represents the most generic scenario for a distributed HC infrastructure that receives many kinds of tasks. Finally, the last consistency category is the one of the *semi-consistent* ETC scenarios, this category model those inconsistent systems that include a consistent subsystem.

The second considered model, the EMC model, was proposed by Nesmachnow et al. (2012a). The EMC is a novel model for multi-core energy-aware HC systems which considers both the computing time and the energy consumption, hence it is very suitable for modeling the ME-HCSP. The model tackles two important shortcomings of previously defined methods: (a) it accounts for the energy consumption in the system, and (b) it models multi-core machines, a technology which nowadays is present in almost every computing system.

The EMC model separately models the computing resources (or *machines*) of the HC system, and the task workload (or just *tasks*). Each machine in the model is represented by the following attributes: *i) computing power* (*op*) which represents the number of operations a machine is able to compute in a time unit, defined by taking into account the *op* value is reported by the SSJ benchmark from the Standard Performance Evaluation Corporation (SPEC); *ii)* the *number of processing cores* (*cores*) of a given machine which is the number of parallel processing cores available in that machine; *iii)* the *minimum energy consumption* (E_{idle}) which is the energy consumption of a machine each time unit when the machine is idle; and *iv)* the *maximum energy consumption* (E_{max}) which is the energy consumption of a machine each time unit when the machine is processing a task.

Two methods are proposed for defining execution time requirements of a task in a machine, the *related* and the *unrelated* method. Both methods define each task to require a given fixed number of operations ($TO(t_i)$). The related method simplifies some heterogeneities in the system and defines the execution time of a task t_i to be directly proportional to the computing power of the machine m_j in which it is executed, $ET(t_i, m_j) = TO(t_i)/[op(m_j)/cores(m_j)]$. On the other hand, the unrelated method includes an additional machine-specific cost deviation ($AO(t_i, m_j)$) which represents external overhead affecting the machine (such as networking access overhead, memory limitations, etc.). Considering this external factors, the unrelated method defines the execution time of a task t_i in a machine m_j as $ET(t_i, m_j) = [TO(t_i) + AO(t_i, m_j)]/[op(m_j)/cores(m_j)]$.

Task workload scenarios are classified in three different heterogeneity categories, being *low* heterogeneity, *medium* heterogeneity, and *high* heterogeneity.

Instead of synthetically generating the machine scenarios, the EMC model proposes the generation of the scenarios based on a list of surveyed hardware which includes 64 nowadays CPUs. Table 2.1 describes the surveyed hardware.

Both the ETC model and EMC model assume the availability of complete knowledge regarding the execution time of the tasks in the HC system. Unfortunately, it is not practical to empirically determine the execution time of each task in each machine in a HC system hence estimation methods are needed in order to approximate the different execution times of a given task in the HC system. In this regard, studies have shown that tasks execution time prediction techniques present accurate results which can be used as inputs of a HC system scheduling algorithm (Bohlouli and Analoui, 2008; Glasner and Volkert, 2009; Li et al., 2004; Seneviratne and Levy, 2011).

2.3.3 Problem instances

Ali et al. (2000) introduced the ETC model and proposed two different methods for generating ETC problem scenarios, however no standard set of scenarios for the HCSP were provided in his work. Later, Braun et al. (2001), using the ETC model introduced by Ali et al. (2000), did generate some scenarios for the HCSP which where used in many related works (Aggarwal et al., 2005; Kołodziej et al., 2011; Ritchie and Levine, 2003; Xhafa et al., 2008b). The set of scenarios generated by Braun et al. (2001) comprise 12 scenarios with every possible combination of task heterogeneity, machine heterogeneity, and consistency, and with 512 tasks and 16 machines each scenario (dimension 512×16). However, in the last decade, HC systems have considerably grown in size (Agarwal et al., 2007; Hey and Trefethen, 2002; Jones, 2005; Shiers, 2007; Wilkins-Diehr et al., 2008), and the benchmark instances generated by Braun et al. (2001) no longer model real-sized scenarios. Because of this, Nesmachnow et al. (2010) generated an updated set of benchmarking scenarios for the HCSP ranging from *small* sized HC system scenarios including dimensions up to 1024×32; *medium* sized HC system scenarios with dimensions up to 4096×128; and *large* sized HC system scenarios with dimensions up to 8192×256.

Nesmachnow et al. (2012a) also generated a set of EMC scenarios for the ME-HCSP. In their work, Nesmachnow et al. (2012a) provide a set of scenarios with dimensions of 512×16, 1024×32, and 2048×64. A total of 20 machine scenarios and 40 task workloads scenarios are provided for each scenario dimension, totaling a number of 800 ME-HCSP different scenarios for each dimension.

Table 2.1: Hardware considered in the EMC model

	processor	GHz	cores	op	E_{idle} (W)	E_{max} (W)	$^{op}/_{Watt}$
1	AMD Opteron 2216 HE	2.4	2	47927	82	138	101.5
2	AMD Opteron 2356	2.3	4	122114	79.3	150.3	521.5
3	AMD Opteron 2376 HE	2.3	4	173163	59.5	105	1044
4	AMD Opteron 2377 EE	2.3	4	171052	36.6	86	1379
5	AMD Opteron 2380	2.5	4	154045	69	134.5	731
6	AMD Opteron 2382	2.6	4	165629	63.5	129	851
7	AMD Opteron 2384	2.7	4	169068	68.5	131.6	834.5
8	AMD Opteron 2419 EE	1.8	6	203477	37.3	89	1614
9	AMD Opteron 2425 HE	2.1	6	231108	40.2	109	1532
10	AMD Opteron 2435	2.6	6	256780	61	129.5	1350.5
11	AMD Opteron 4164 EE	1.8	6	211874	30.7	67.8	2043
12	AMD Opteron 6168	1.9	12	416178	48	131	2210
13	AMD Opteron 6174	2.2	12	459729	40.5	134.7	2481.1
14	AMD Opteron 6176	2.3	12	469103	42.8	140.7	2452.3
15	AMD Opteron 6262 HE	1.6	16	396768	35.2	106.5	2618
16	AMD Opteron 6276	2.3	16	570390	35.8	152.8	2868.3
17	AMD Opteron 8376 HE	2.3	4	153377	58	105.7	938.3
18	Intel Core i3-540	3.07	2	136634	25	61.7	1556
19	Intel Core i7 610E	2.53	2	126976	20.8	47.4	1856
20	Intel Pentium D 930	3	2	52303	105	169	190
21	Intel X3350	2.67	4	173021	68.9	119	913
22	Intel Xeon 3040	1.86	2	54479	86	117	268
23	Intel Xeon 3075	2.66	2	98472	93.7	135	431
24	Intel Xeon 5160	3	2	82084	101.8	151.5	328
25	Intel Xeon 7020	2.66	2	21521	130	208.3	61.1
26	Intel Xeon 7110M	2.6	2	37185	143.8	183	114
27	Intel Xeon E3110	3	2	118486	75.2	117	605
28	Intel Xeon E5345	2.33	4	119239	114	170.8	413
29	Intel Xeon E7330	2.4	4	126597	104.5	156.8	489
30	Intel Xeon L3360	2.83	4	183767	48.7	95	1253
31	Intel Xeon L5335	2	4	111872	107.3	146	443
32	Intel Xeon 3070	2.67	2	78928	78.8	120	405
33	Intel Xeon 5160	3	2	79576	86	129	382
34	Intel Xeon E3-1220	3.1	4	329862	22.4	92.8	3026
35	Intel Xeon E3-1260L	2.4	4	314438	17.8	55.7	4327.5
36	Intel Xeon E3-1270	3.4	4	394356	21.4	102	3265
37	Intel Xeon E3-1280	3.5	4	412077	30.9	106.2	3214
38	Intel Xeon E5420	2.5	4	143516	77.5	130	681
39	Intel Xeon E5440	2.83	4	150979	76.9	131.8	709
40	Intel Xeon E5462	2.8	4	161999	65	126	854
41	Intel Xeon E5472	3	4	137497	92.7	160.3	551.3
42	Intel Xeon E5540	2.53	4	275475	34.5	110.3	1807.5
43	Intel Xeon E7-4870	2.4	10	556330	105.8	209.3	1816
44	Intel Xeon E7-8870	2.4	10	602771	126.5	246.8	1586
45	Intel Xeon L3360	2.83	4	199707	55.2	101.9	1255.6
46	Intel Xeon L5335	2	4	100845	93.5	130	446
47	Intel Xeon L5408	2.13	4	121036	104.2	136	505
48	Intel Xeon L5410	2.33	4	144002	64.1	105	838
49	Intel Xeon L5420	2.5	4	138462	63.3	105.1	835.1
50	Intel Xeon L5430	2.67	4	152533	68.9	108.8	870.9
51	Intel Xeon L5520	2.26	4	228603	31.8	89.5	1791.5
52	Intel Xeon L5530	2.4	4	251057	32.3	89.5	1981
53	Intel Xeon L5630	2.13	4	226022	30.4	69	2118
54	Intel Xeon L5640	2.27	6	318441	35.7	96.1	2562.6
55	Intel Xeon L7345	1.86	4	89881	67.8	96.8	546
56	Intel Xeon X3220	2.4	4	143742	79.8	132	667
57	Intel Xeon X3360	2.83	4	188173	59.6	118.5	1343.8
58	Intel Xeon X3470	2.93	4	307615	38.2	120	2051.2
59	Intel Xeon X3480	3.07	4	325650	38.3	121.5	1912.9
60	Intel Xeon X5272	3.4	2	104222	93	140	450
61	Intel Xeon X5570	2.93	4	280253	41.2	122.8	1873.8
62	Intel Xeon X5670	2.93	6	443655	37.2	126.1	2656.4
63	Intel Xeon X5675	3.07	6	459263	59.9	150.4	2426.2
64	Intel Xeon X7560	2.27	8	461606	135.6	241.5	1239.6

2.4 Algorithms for solving the heterogeneous computing scheduling problem

Scheduling independent non-preemptive jobs onto parallel machines to minimize the makespan is one of the most tackled problems in scheduling theory. The aforementioned problem is already strongly NP-hard when the jobs are to be scheduled in m identical parallel machines (Leung et al., 2004), $P||C_{max}$ in Graham et al. (1979) notation, hence its not feasible to compute exact solutions for practical problem scenarios.

Some algorithms and techniques available in the literature for solving scheduling related problems are presented in the following sections.

It is important to note that most of the following methods tackle the HCSP, and not the ME-HCSP. This is because the classic version of the problem do not considers energy consumption as an objective. In fact, the study of the ME-HCSP is new, and this thesis is a significant contribution to the development and improvement of energy-saving mechanism in HC systems.

2.4.1 Enumerative algorithms

Enumerative algorithms find exact solutions through enumerative search. Enumerative search does not imply brute force search; usually enumerative algorithms include some elimination rules in order to prune the search space and reduce the computational cost of the algorithm. A number of enumerative algorithms were proposed for solving the HCSP problem.

A brand and bound algorithm based on surrogate relaxation and duality was proposed by van de Velde (1993) for the HCSP solving dimensions up to 60×20 in about 5 minutes execution time. Martello et al. (1997) also proposed a branch and bound algorithm for the HCSP solving scenarios with dimensions up to 200×5. In a more recent work, Salem et al. (2000) also designed a branch and bound algorithm to solve HCSP scenarios with dimension up to 40×8 in reasonable time.

2.4.2 Linear programming based algorithms

Any NP-hard combinatorial optimization problem can be seen as an easy-to-solve problem complicated by a number of nasty side constraints (van de Velde, 1993). A candidate strategy for solving NP-hard problems is to relax those nasty side constraints and tackle the easier to solve resulting relaxed problem. A common relaxation in MILP problems consists in applying a *linear programming relaxation* which consists in replacing the integrality constraints with weaker non-integer conditions so the resulting linear programming (LP) problem is solvable in polynomial time. This way, near-optimal solutions for the HCSP problem can be computed by rounding optimal LP relaxations.

Lenstra et al. (1990) proposed a polynomial time 2-competitive linear programming relaxation based algorithm, and further shows that there is no approximation algorithm better than 3/2-competitive for the HCSP ($R||C_{max}$). The previous approach was extended by Shmoys and Tardos (1993) introducing an improved rounding technique and showing that there is a polynomial time 2-competitive approximation algorithm to simultaneously minimize the weighted completion time sum and the makespan objectives.

2.4.3 List-scheduling algorithms

Several polynomial time approximation heuristics have been proposed for solving the HCSP. One of the most used family of such methods is the one of list-scheduling heuristics which can be characterized as a class of *constructive heuristics* (Kwok and Ahmad, 1999). Constructive heuristics generate solutions from scratch by adding predefined solution components to an initially empty solution. This is done until a solution is complete or some other stopping criteria is satisfied (Blum and Roli, 2008). List-scheduling methods work by assigning priorities to tasks based on a particular criteria, sorting the list of tasks by priority and assigning each task to a machine in decreasing order until all tasks are assigned. There are many list-scheduling heuristics devised to solve the HCSP, some of them are detailed next.

- *Minimum Completion Time* (MCT) considers the set of tasks sorted in an arbitrary order. Then, it assigns each task to the machine with the minimum execution time for that task (Braun et al., 2001).

- *MinMin* 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 machine that executes it faster. The mapped task is removed from U, and the process is repeated until all tasks are mapped (Ibarra and Kim, 1977).

- *MaxMin* is very similar to MinMin. 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 maximum overall MCT to the machine that executes it faster. The mapped task is removed from U, and the process is repeated until all tasks are mapped (Ibarra and Kim, 1977).

- *Duplex* is a combination of the MinMin and MaxMin heuristics. The Duplex heuristic performs both MinMin and MaxMin heuristics, and then uses the best solution (Freund et al., 1998).

- *Sufferage* identifies the task that, if not assigned to a certain host, will *suffer* the most. The *Sufferage value* is computed as the difference between the best MCT of the task and its second-best MCT, and this method gives precedence to those tasks with high Sufferage value. Then, it assigns them to the machines that can complete these tasks at the earliest time (Maheswaran et al., 1999).

- *Longest Job to Fastest Resource-Shortest Job to Fastest Resource* (LJFR-SJFR) initially assigns a number of the longest tasks, equal to the number of available machines, to the fastest available machines (application of the LJFR heuristic). Then, for each remaining tasks the LJFR heuristic or SJFR heuristic are applied alternatively. The SJFR assigns the shortest task to the fastest available machine (Xhafa and Abraham, 2008).

Furthermore, two variants of the previously defined heuristics have been used in this work to generate initial solutions for the algorithms proposed in this work in order to solve the ME-HCSP:

- *Randomized Minimum Completion Time* (rMCT) is a randomized version of MCT, randomly sorting the task collection before applying the MCT algorithm.

- *pMinMinDD* or parallel MinMin with domain decomposition, is an multi-threading version of the MinMin algorithm. It applies a domain decomposition strategy splitting the tasks domain T into n equally sized sub domains $T_1, ..., T_n$, then it computes the solution for the n subproblems $T_i \times P$ applying the Min-Min heuristic and computing n sub-schedules $f_1, ..., f_n$. Finally, the n sub-schedules are aggregated into the final schedule $f = f_1 \cup ... \cup f_n$. The implementation makes use of a pool of threads in order to compute each sub-domain schedule in parallel. (Canabé and Nesmachnow, 2012).

The *rMCT* algorithm provides diversity to the classic *MCT* algorithm and helps to avoid some ill conditioned scenarios due to some arbitrary task ordering. The *pMinMinDD* algorithm tries to alleviate the fact that the *MinMin* algorithm has a $O(n^3)$ execution order; because of its cubic execution order, MinMin can be non-practical when solving very large problem scenarios.

2.4.4 Metaheuristic algorithms

As defined by Gendreau and Potvin (2010), metaheuristics are solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space. Broadly speaking, metaheuristics methods can be characterized as approximate iterative methods designed to solve optimization problems making few or no assumptions about the problem being solved.

There are many metaheuristics methods available in the literature. Some of them can be characterized in the following classes (Brownlee, 2011): *i) Local Search*-based methods such as Stochastic Hill Climbing (Forrest and Mitchell, 1992), Iterated Local Search (ILS) (Stützle, 1999), Guided Local Search (Voudouris, 1998), Greedy Randomized Adaptive Search (GRASP) (Feo and Resende, 1989), Tabu Search (TS) (Glover, 1986), etc.; *ii) Evolutionary Algorithms* based in the evolution of the species such as Generic Algorithm (GA) (Goldberg, 1989), Genetic Programming (Koza, 1992), Evolution Strategies (Rechenberg, 1973), Differential Evolution (Storn and Price, 1997), Memetic Algorithm (MA) (Moscato, 1989), etc.; *iii) Physical Algorithms* inspired in physical processes such as Simulated Annealing (SA) (Kirkpatrick et al., 1983), Gravitational Search Algorithm (GSA) (Rashedi et al., 2009), etc.; *iv) Swarm Algorithms* inspired in the collective intelligence of swarms such as Particle Swarm Optimization (PSO) (Eberhart and Kennedy, 1995), Ant Colony (ACO) (Dorigo and Gambardella, 1997), etc.

A number of these metaheuristics have been used for solving scheduling problems in HC systems. Abraham et al. (2008) proposed various metaheuristics for solving the HCSP: GA, SA, PSO, and ACO algorithms for scenarios with dimensions up to 1000×100 were tackled. Izakian et al. (2009b) devised a PSO-based algorithm for solving the HCSP, minimizing the makespan problem objective, using the ETC model proposed by Braun et al. (2001), and solving the scenarios dimension of up to 1024×16. Most metaheuristics work by iteratively improving a starting solution, sometimes called *initial* solution, thus many applied metaheuristics make use of some constructive heuristic. In the work by Izakian et al. (2009b), the MinMin list-scheduling algorithm was used in order to compute an initial solution which was used as input for the PSO-based algorithm. The results computed with the proposed PSO-based algorithm were compared against the results computed by a pure PSO algorithm and a GA. More recently, Subashini and Bhuvaneswari introduced a GA-based and a PSO-based algorithms for solving the HCSP (Subashini and Bhuvaneswari, 2011a,b), both considering the simultaneous minimization of the makespan and the total sum of the completion time of each task (*flowtime*), using the ETC model and solving the scenarios proposed by Braun et al. (2001).

Krömer et al. (2009) designed a differential evolution algorithm for the HCSP in order to minimize makespan and flowtime using the ETC computing model. Experiments were conducted tackling the scenarios proposed by Braun et al. (2001), and a set of list-scheduling heuristics such as MaxMin, Sufferage, and MinMin were used to compute initial solutions for the problem. Later, Krömer et al. (2011) also presented a differential evolution algorithm for the HCSP implemented in Graphic Processing Units (GPU). Again it used the ETC model and solved the instances proposed by Braun et al. (2001). With the additional computing power provided by the GPU, Krömer et al. (2011) was able to improve the results computed in Krömer et al. (2009).

Nesmachnow and Iturriaga (2011) proposed a CHC evolutionary algorithm hybridized with a local search algorithm for solving the HCSP with the additional consideration of some economic related quality of service metric. The ETC model was adopted, the MCT, MinMin, and Sufferage algorithms were used to compute initial solutions, and scenarios with dimensions up to 4096×128 were tackled. Nesmachnow et al. (2012b) proposed a local search for solving the HCSP using the ETC model and solving instances with dimensions up to 4096×128 in less than 6 seconds. Pinel et al. (2013) introduced a cellular GA implemented for the GPU for solving the HCSP in order to minimize the makespan objective using the ETC model. In his work, Pinel et al. (2013) also presented a implementation of the MinMin algorithm for the GPU and used it to compute some initial solutions for the problem. With the proposed algorithms, Pinel et al. (2013) was able to solve scenarios with dimensions up to 65536×2048.

2.5 Summary

This chapter defined the term *heterogeneous computing system* and presented the problem of scheduling independent tasks in energy-aware HC systems. The ME-HCSP was mathematically defined and characterized in the general scheduling problem notation proposed by Graham et al. (1979). The EMC model proposed by Nesmachnow et al. (2012b) and a set of real-world sized scenarios were reviewed and will be adopted in the following chapters in order to tackle the ME-HCSP. Finally, a small survey of different methods to tackle the HCSP is presented. First, some exact enumerative algorithms are reviewed. Then a collection of polynomial time approximation methods are presented: some based on linear programming relaxations, some based on list-scheduling algorithms, and last, some single-objective and multi-objective non-deterministic metaheuristics are introduced.

Next, in chapter 3, a comprehensive introduction to metaheuristics methods is presented. After that, in Chapter 4, a complete survey of recent metaheuristic methods for solving the ME-HCSP is detailed.

Chapter 3

Metaheuristic algorithms

This chapter introduces metaheuristic algorithms. First, some general notions about metaheuristic algorithms are presented and a classification criterion for metaheuristic algorithms is detailed. Then, stochastic optimization fundamentals are described and a general framework for a stochastic search method is reviewed. After that, local search algorithms are introduced as one of the most successful general approaches when tackling hard combinatorial optimization problems. Some notions of multi-objective optimization are discussed along with the increased complexity when dealing with more than one objective function. Two well-known metaheuristic algorithms are described to exemplify the main features of single-objective and multi-objective optimization: the Iterated Local Search (ILS) (Stützle, 1999) and the Pareto Archived Evolution Strategy (PAES) (Knowles and Corne, 2000).

3.1 Introduction

The concept of metaheuristic algorithm was initially introduced by Glover (1986). Broadly speaking, metaheuristics are methods for solving optimization problems but there is no commonly accepted definition for the term metaheuristic. According to Osman and Laporte (1996),"a metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the *search space*, learning strategies are used to structure information in order to find efficiently near-optimal solutions" for an optimization problem. The *search space* of an optimization problem is the space defined by all the feasible solutions of that optimization problem.

The following are some fundamental properties which characterize metaheuristics (Blum and Roli, 2003): *i)* metaheuristics are strategies that guide the search process; *ii)* the goal is to efficiently explore the search space in order to find near-optimal solutions; *iii)* techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes; *iv)* metaheuristic algorithms are approximate and usually non-deterministic; *v)* they may incorporate mechanisms to avoid getting trapped in confined areas of the search space; *vi)* the basic concepts of metaheuristics permit an abstract level description; *vii)* metaheuristics are not problem-specific; *viii)* metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.

There are many ways to classify metaheuristic algorithms depending on their characteristics. One of the most widely used classification criterion is based on whether the metaheuristic is *population-based*, or *trajectory-based* (Blum and Roli, 2008). This classification takes into account the number of solutions being considered at the same time by the algorithm. Algorithms iteratively improving one solution at a time are called *trajectory methods*, such as: Tabu Search (TS), Greedy Randomized Adaptive Search Procedure (GRASP), Iterated Local Search (ILS), Variable Neighborhood Search (VNS), etc. On the other hand, *population-based* algorithms describe an iteration over a set of solutions taking all of them into account simultaneously. Some examples of such methods are: Genetic Algorithm (GA), Evolutionary Strategy (ES), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), etc.

Other classification criteria are *nature-inspired metaheuristics vs. non-nature inspired*, based on the origins of the metaheuristic; or *memory-based vs. memory-less methods*, based on whether the method use some kind of learning technique of if it just use the current state of the search in order to determine the next action; etc.

3.2 Stochastic search

Stochastic search is a crucial technique used on most metaheuristics. Not every metaheuristic applies a non-deterministic approach; however, the incorporation of pseudo-random behavior into a metaheuristic algorithm is rather the usual than the exceptional (Blum and Roli, 2003). According to Glover (1986):

> "In the face combinatorial complexity, there must be freedom to depart from the narrow track that logic single-mindedly pursues, even on penalty of losing the guarantee that a desired destination will ultimately be reached. ('Ultimately' may be equivalent to 'a day before doomsday' from a practical standpoint.)
>
> In brief, effective strategies for combinatorial problems can require methods that formal theorems are unable to justify. That is not to say such methods lack 'structure'. Complete flexibility has its parallel in complete randomization, which is scarcely a useful basis for complex problem solving. Methods that are 'intelligently flexible' lie in some yet-to-be-defined realm to which theorems (of the familiar sort) do not apply, yet which embraces enough structure to exclude aimless wandering."

Most stochastic optimization methods consist of two different phases (Telega, 2007): *i)* a *global phase* or *exploration phase*, where the objective function is evaluated in different sections of the search space in order to identify promising sections of the space which might contain near-optimal solutions; and *ii)* a *local phase* or *exploitation phase*, where the previously identified promising sections are thoroughly searched in order to yield one or more near-optimal solutions.

Although stochastic methods do not have strict convergence guarantees, in many cases they have proven to compute accurate solutions, and are practical methods to tackle problems that are (currently or forever) out of the reach of theoretically correct, rigorous methodology (Pardalos and Romeijn, 2002). Because of the lack of strict convergence guarantees, one of the capital questions when applying a stochastic search method, is when to stop searching for better solutions to avoid the wasting of computational resources. Many methods were proposed in the literature for determining the *stopping criterion* (Safe et al., 2004). These methods can be classified into: *static stopping criteria* or *dynamic stopping criteria.*

Static stopping criteria are stopping criterion that do not change during the progression of an iterative optimization algorithm, e.g. timespan stopping criterion or iteration count stopping criterion. In this regard, studies where performed for some metaheuristics in order to find upper bounds for the number of iterations necessary to ensure finding an optimal solution with a prescribed probability (Studniarski, 2010).

On the other hand, the *dynamic stopping criteria* or *online stopping criteria* are techniques which aim to analyze different progress indicators during the progression of an iterative optimization algorithm in order to detect the convergence of the algorithm, when this happens the algorithm is stopped. Convergence detection is not trivial matter when dealing with multiple objectives, hence several progress indicators were proposed in this regard (Wagner et al., 2011).

A general framework for describing a stochastic search method was presented by Gendreau and Potvin (2010). In this framework, $\mathcal{S}$ is a finite search space and f is the objective function. A stochastic search method is an iterative algorithm which in iteration t, uses a memory M_t and a list L_t of solutions $s \in \mathcal{S}$. The list L_t contains new *candidate solutions* for the optimization problem, and the list L_t^+ contains the pairs $(s, f(s)), \forall s \in L_t$. The algorithm proceeds as shown in Algorithm 1.

Algorithm 1 General framework for the stochastic search method

```
1: initialize M_1 according to some rule
2: for iteration t = 1, 2, ..., until some stopping criterion is satisfied do
3:    determine list L_t as a function g(M_t, z_t) of M_t of random influence z_t
4:    determine objective function values f(s) of all s ∈ L_t and form a list L_t^+ containing
      the pairs (s, f(s))
5:    determine new memory content M_{t+1} as a function h(M_t, L_t^+, z'_t) of the current
      M_t, of the list of solution-value pairs L_t^+, of random influence z'_t
6: end for
```

The general stochastic search method starts by generating an initial solution, e.g. using some constructive heuristic. Then, the method iterates until some stopping criterion is met. The stopping criterion defined in line 2 can depend on (M_t, L_t^+) if the algorithm make use of a dynamic stopping criterion. In this formalism, z_t and z'_t represents vectors of pseudo-random numbers that are used by the stochastic algorithm. The function $g(M_t, z_t)$ in line 3 specifies the list of new candidate solutions, and the function $h(M_t, L_t^+, z'_t)$ in line 5 specifies the new content of the memory.

The functions $g(M_t, z_t)$ and $h(M_t,\ L_t^+,\ z'_t)$ may use any information on the problem instance. The important special case where g and h are only allowed to use the knowledge of the search space $\mathcal{S}$ and of the problem type, but not the knowledge of the concrete problem instance, is denoted as *black-box optimization* (Droste et al., 2006). The black-box optimization approach is the most interesting formulation since it provides a generic approach for solving the optimization problem.

3.3 Local search methods

Local search algorithms are one of the most successful general approaches for finding high quality solutions for hard combinatorial optimization problems in reasonable time (Hoos and Stützle, 2004).

According to Blum and Roli (2008), in an optimization problem where $\mathcal{S}$ is the search space of the problem, the neighborhood structure of a solution is defined as follows.

Definition 1. A **neighborhood** structure is a function $\mathcal{N} : \mathcal{S} \to 2^{\mathcal{S}}$ that assigns to every $s \in \mathcal{S}$ a set of neighbors $\mathcal{N}(s) \subseteq \mathcal{S}$. $\mathcal{N}(s)$ is called the neighborhood of s.

According to the previous definition, solution s' is a neighbor of the solution s if s' can be obtained by applying the function $\mathcal{N}$ to s, that is $s' \in \mathcal{N}(s)$. Often, neighborhood structures in combinatorial optimization problems are implicitly defined by specifying the changes that must be applied to a solution s in order to generate all its neighbors. A well-known neighborhood structure in combinatorial optimization problems is the *k-exchange neighborhood* (Aarts and Lenstra, 1997). The *k-exchange neighborhood* can be defined as follows: given a solution represented as a sequence, the *k-exchange neighborhood* is the neighborhood obtained by exchanging k elements in the sequence.

A local search algorithm starts from some initial solution s and iteratively tries to replace the current solution s by a better solution s' in an appropriately defined neighborhood $\mathcal{N}(s)$ of the current solution. At each iteration, a local search procedure performs a search for a candidate solution s' within the neighborhood $\mathcal{N}(s)$, and evaluates the various neighboring solutions. The procedure either accepts or rejects the candidate solution s' as the next schedule to move to, based on a given acceptance-rejection criterion. A high level local search algorithm is sketched in Algorithm 2.

Algorithm 2 Pseudo-code of a local search algorithm

```
1: s ← GenerateInitialSolution()
2: while stopping criterion is not satisfied do
3:   s' ← SearchNeighbor(N(s))
4:   if s' satisfies acceptance criterion then
5:     s ← s'
6:   end if
7: end while
8:
9: return s
```

The local search algorithm starts by generating an initial solution. The fastest way for generating an initial solution is often to generate it at random, but—if possible— the initial solution should be a quality solution for the problem. Because of this, function *GenerateInitialSolution()* in line 1 usually makes use of a constructive method, such as the list-scheduling heuristic methods presented in Section 2.4.3 for the HCSP, in order to generate a feasible solution from scratch.

After the initial solution is generated, the algorithm starts to iterate until some stopping criterion is met. Each iteration, the algorithm first searches the current neighborhood using the function *SearchNeighbor($\mathcal{N}(s)$)* in line 3. The function *SearchNeighbor($\mathcal{N}(s)$)* can be implemented in several ways, being the two most typical ones: *first-improvement* or *best-improvement*. A *first-improvement* function searches the neighborhood $\mathcal{N}(s)$ and returns the first solution that is better than s. On the other hand, a *best-improvement* function exhaustively explores the neighborhood and returns the best solution in the neighborhood. If the solution s' found by the function *SearchNeighbor($\mathcal{N}(s)$)* satisfies some acceptance criterion, then the solution s is replaced by the new solution s'.

Both the definition of the neighborhood structure $\mathcal{N}$ and the implementation of the *SearchNeighbor($\mathcal{N}(s)$)* are critical issues in the design of a local search.

Generally, the larger the neighborhood structure the more solutions are explored in each iteration, potentially leading to better quality solutions and improving the chances of finding a better final solution. On the other hand, the larger the neighborhood the longer it will take to compute the search in the neighborhood structure each iteration. This additional computing cost may impact prohibitively in the execution time of the algorithm when solving real-world scenarios (Gendreau and Potvin, 2010).

3.3.1 An example of a local search based metaheuristic: the ILS algorithm

Iterated Local Search (ILS) is a well-known metaheuristic originally proposed by Stützle (1999) which iteratively builds a sequence of solutions generated by an embedded local search heuristic with a multi-start technique. Multi-start techniques were proposed as a way to include some exploring capabilities to neighborhood search methods by simply applying the search methods from multiple random initial solutions. In the ILS algorithm, the multi-start technique is applied making use of a random perturbation mechanism. The perturbation mechanism is applied to the current solution before applying the embedded local search heuristic with the hope of: *i)* re-starting the local search from a quality solution and *ii)* escaping the basin of attraction of the local minimum where the current solution is. Algorithm 3 shows the pseudo-code of the ILS algorithm (Hoos and Stützle, 2004).

Algorithm 3 Pseudo-code of the ILS algorithm

```
1: s ← GenerateInitialSolution()
2: s ← LocalSearch(s)
3: while stopping criterion not met do
4:    s′ ← Perturbation(s)
5:    s′ ← LocalSearch(s′)
6:    if AcceptanceCriterion(s, s′) then
7:       s ← s′
8:    end if
9: end while
10:
11: return s
```

The ILS algorithm starts by generating an initial solution s (line 1). The embedded local search procedure is applied to the generated solution s in order to obtain a locally optimal solution. After that, the main iteration loop is repeated until the stopping condition in met (lines 3–8). Each iteration of the main loop consists of three major stages: *i)* a random perturbation is applied to the current solution s, obtaining a perturbed solution s' (line 4); *ii)* the embedded local search is applied to the perturbed solution s', obtaining a new locally optimal solution (line 5); finally, *iii)* an acceptance criterion is used in order to determine from which of the two local optima the process continues, s or s' (lines 6–8).

The perturbation mechanism is a key element in the ILS schema. If the perturbation does not perturb enough the current solution, the local search might not be able to escape the basin of attraction of the local minimum where the current solution is. On the other hand, if the perturbation is too strong, the perturbed solution might lose all its quality properties and the procedure would be similar to a random re-start. In order to deal with this problem, in the ILS algorithm both the $Perturbation$ and the $AcceptanceCriterion$ functions, may maintain a search history in order to dynamically adapt the algorithm during the search. For example, if the same local optima is repeatedly encountered, the $Perturbation$ function could apply stronger perturbations, and the $AcceptanceCriterion$ function could be more prone to accept worse quality solution, in order to escape the local minimum which is stagnating the search.

3.4 Multi-objective optimization

Many real world optimization problems have two or more objectives which are to be optimized simultaneously. In these *multi-objective optimization problems* (MOP), some objectives are usually in conflict with each other, hence improving the quality of one objective means degrading the quality of at least one of the other objectives (Coello et al., 2010). In a single-objective optimization problem, the optimization goal is represented by a unique objective function. On the other hand, in a MOP the optimization goal is represented by a vector of objective functions, each objective described by a different function in the vector of functions. The most general formulation for a multi-objective optimization problem is defined in Equation 3.1.

$$\begin{aligned} &\text{optimize } F(x) = (f_1(x), ..., f_n(x)) \\ &\text{with } x \in \mathcal{S} \end{aligned} \tag{3.1}$$

In the previous formulation, n is the number of objectives, x is a vector of decision variables of the form $x = (x_1, ..., x_k)$, $\mathcal{S}$ is the search space of feasible solutions, and each function $f_i(x)$ is an objective function to be minimized or maximized.

Since some objectives in a MOP are usually in conflict, there is not a single best solution as in a single-objective optimization problem. Optimizing a MOP involves finding several solutions, called *Pareto optimal solutions*, which represent the best possible trade-off values among all the MOP objectives.

When considering the most general case in which all objectives are equally important it is assumed that a solution s is preferable to another solution s' only if s is at least as good as s'. Then, the best trade-off solutions must be considered equally good solutions since their representing vectors cannot be completely ordered. Because in this formulation the preference for one *Pareto optimal solution* cannot be asserted over the other Pareto optimal solutions, the decision of determining which solution to choose as the single final solution of the considered MOP must not be part of the problem solving. The problem solving method should compute as many Pareto optimal solutions as possible and leave the choosing of the final solution to an external *decision making procedure* (often a human decision maker) (Coello et al., 2006; Deb, 2001). In order to define the concept of Pareto optimal solution, first the concept of dominance between solutions must be defined.

Definition 2. It is said that a solution $y = (y_1, ..., y_k)$ **dominates** a solution $z = (z_1, ..., z_k)$, in a minimization context, if and only if $\forall i \in [1...n], f_i(y) \leq f_i(z)$ and $\exists i \in [1...n]$ such that $f_i(y) < f_i(z)$.

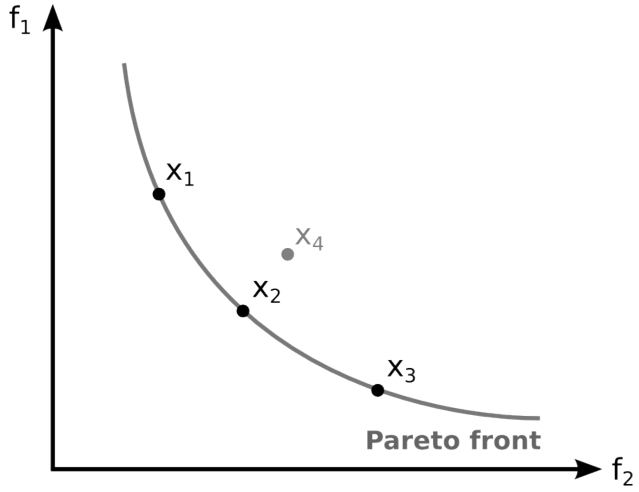


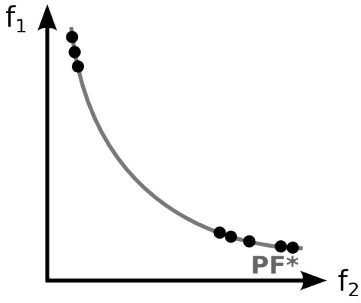
Figure 3.1: Pareto front representation.

Definition 3. A solution $x^* \in \mathcal{S}$ is a **Pareto optimal solution** (or non-dominated solution) if and only if there is no solution $x \in \mathcal{S}$ such that x dominates x^*.

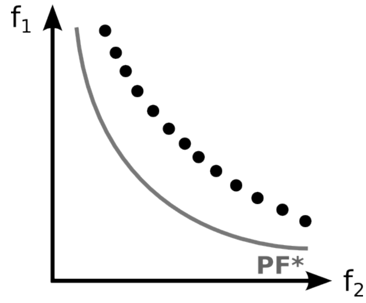
The set all of the Pareto optimal solutions of a MOP is named *Pareto optimal set* (P^*), and the set of the objective functions values defined by the Pareto optimal set in the function space is known as the *Pareto optimal front* (PF^*). Approximation methods, such as metaheuristic algorithms, are not expected to compute the *Pareto optimal set* when solving a MOP. Instead, an approximation algorithm will compute a *Pareto approximation set* (P), which is not the optimal set of trade-off solutions for the problem, but the best trade-off solutions the algorithm could find. This P set projected into the function space is known as the *Pareto approximation front* (PF). It is worth noting that even though there is only one *Pareto optimal set* for a problem, there are many *Pareto approximation set*, one for each approximation method the problem is tackled with.

Figure 3.2 represents a Pareto front of a bi-objective minimization problem. In this figure solutions x_1, x_2, and x_3 belong to the set P, but solution x_4 does not. It can be shown that neither x_1, x_2, nor x_3 are dominated by any other solutions, that is $\nexists j \in \mathcal{S}$ such that $f_1(j) \leq f_1(x_i)$ and $f_2(j) \leq f_2(x_i)$, for $i = 1, 2, 3$. For example $f_1(x_2) \leq f_1(x_1)$ but $f_2(x_2) \nleq f_2(x_1)$. On the other hand, solution x_4 is non-dominated with respect to x_1 and x_3, but it is clearly dominated by solution x_2 since $f_1(x_2) \leq f_1(x_4)$ and $f_2(x_2) \leq f_2(x_4)$, hence $x_4 \notin P$.

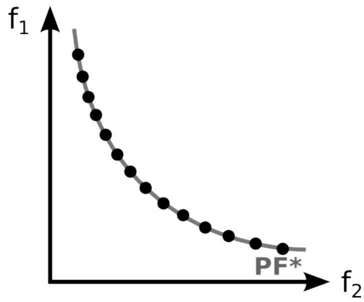
When tackling a MOP, two goals must be considered simultaneously: to compute accurate solutions for every objective function, and to maintain diversity in the computed solutions in order to adequately sample the Pareto front. The former desired goal, to optimize as much as possible every objective function of the MOP, is certainly a desired goal when tackling any optimization problem. The later desired goal is related to the need of having an external *decision making procedure*. When tackling a MOP, usually one does not have *a priori* information about the criteria used by the decision making procedure. Because of this, the set of computed solutions must be as diverse as possible in order to allow the decision making procedure to select an accurate *a posteriori* solution (Deb, 2001). Figure 3.2a and 3.2b show examples of computed solutions not considering both accuracy and diversity as simultaneous goals when tackling a bi-objective minimization problem.



(a) Computed solutions are accurate but diversity is poor



(b) Computed solutions maintain diversity but their accuracy is poor



(c) Computed solutions are accurate and diverse

Figure 3.2: Pareto front examples.

Figure 3.2c present an accurate set of Pareto solutions which are uniformly distributed along the optimal Pareto front of the problem.

In the field of multi-objective optimization problems, metaheuristic methods have become increasingly popular and were applied to solve many real world problems (Nesmachnow et al., 2010). In particular, population-based metaheuristics present the advantage of simultaneously working with a set of possible solutions in each iteration allowing these methods to naturally consider the concept of Pareto solutions set (Coello et al., 2010).

3.4.1 Evaluation metrics for multi-objective optimization

An important task when proposing a method for solving an optimization problem is to be able to compare the results computed by the proposed method with the results computed by other similar methods. Evaluating the results computed by a method for solving a MOP is not an easy task, since the solution to a MOP is a set of trade-off solutions rather than a single value.

Several performance metrics have been proposed for evaluating the computed solutions of a MOP. These metrics can be classified according to whether they are convergence-based metrics (or *efficacy metrics*), diversity-based metrics (or *diversity metrics*), or *hybrid metrics*. (Coello et al., 2010). *Efficacy metrics* provide a measure of accuracy of the computed solution set and the closeness of the obtained Pareto approximation front (PF) with respect to the Pareto optimal front (PF^*). Some examples of *efficacy metrics* are: error ratio (Van Veldhuizen, 1999), generational distance (Van Veldhuizen, 1999), ϵ-indicator (Zitzler et al., 2003), etc.

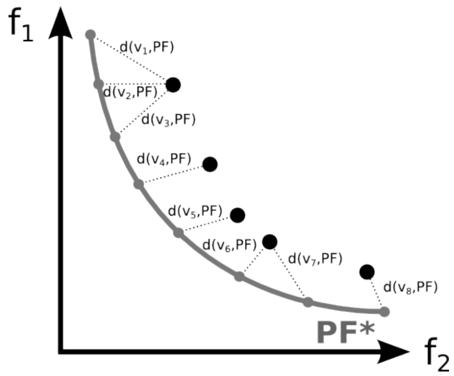


Figure 3.3: Distances computed for the IGD metric ($d(v, PF)$).

Diversity metrics provide a measure of the uniformity of the distribution of the obtained solutions set. Some examples of *diversity metrics* are: spacing (Schott, 1995), spread (Deb, 2001), etc. Finally, *Hybrid metrics* consider both quality and diversity simultaneously. Some examples of *hybrid metrics* are: hypervolume (Zitzler and Thiele, 1998), R-metrics (Knowles and Corne, 2002), etc. The following efficacy metrics will be used in this work during the experimental analysis:

- *Number of non-dominated solutions* (ND). This is one of the most simple quality metric, it accounts for the number of (different) non-dominated solutions found by each algorithm. Clearly, a higher number of non-dominated solutions is better for the diversity measure of the computed solutions set.

- *Inverted generational distance* (IGD). The IGD metric (Zitzler et al., 2003) is defined as the (normalized) sum of the distances between the non-dominated solutions in the Pareto approximation front (PF) and a set of uniformly distributed solutions in the Pareto optimal front (PF^*). The set $\tilde{PF^*}$ is the normalization of the set PF^*, and the set $\tilde{PF}$ is the normalization of the set PF. The formulation of the IGD metric is as follows:

$$IGD = \frac{\sqrt{\sum_{v \in \tilde{PF^*}} d(v, \tilde{PF})^2}}{|\tilde{PF^*}|} \tag{3.2}$$

  In the previous formulation $d(v, PF)$ represents the minimum Euclidean distance between the vector v and the solutions in PF. The closer PF is to PF^*, the lower the value of IGD. Figure 3.3 shows an example all the $d(v, PF)$ distances computed for a bi-objective minimization problem.

The following diversity metric will be used in this work in the experimental analysis:

- *Spread.* The Spread metric was initially proposed by Deb (2001), and is calculated measuring the sum of the relative distance between consecutive solutions in the non-dominated set and also including information about the extreme solutions of the true Pareto front (PF^*). The Spread metric is defined as follows:

$$Spread = \frac{\sum_{m \in M} d_m^e + \sum_{v \in PF} |d_v - \bar{d}|}{\sum_{m \in M} d_m^e + |PF| \times \bar{d}} \tag{3.3}$$

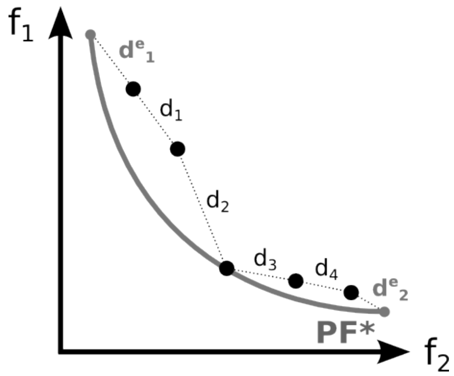


Figure 3.4: Distances to the extremes d^e_m and distances between consecutive solutions d_v, computed for the Spread metric.

In the previous formulation, M is the set of objective functions and d^e_m is the distance between the extreme solutions of PF^* and PF corresponding to the m-th objective function. The distance d_v can be taken as the consecutive Euclidean distance between the i-th and the $(i+1)$-th solutions in the PF set. Finally $\bar{d}$ is the mean of all of the d_v distances. Figure 3.4 shows the distances to the extremes d^e_m and the distances between consecutive solutions d_v, computed for the Spread metric for a bi-objective minimization problem.

Smaller values of Spread mean a better distribution of non-dominated solutions in the calculated Pareto front.

Finally, the *relative hypervolume* hybrid metric is also used during the experimental analysis. The *relative hypervolume* metric is defined as follows:

- *Relative hypervolume* (RHV). The relative hypervolume was initially proposed by Zitzler and Thiele (1998) and calculates the ratio of the volume covered by the PF set and the PF^* set for problems where all objectives are to be minimized. Lets first define hypervolume (HV). For each solution $i \in PF$, a hypercube v_i is constructed with a reference point W and the solution i as the diagonal corners of the hypercube. The reference point W does not have to be a feasible solution, and can simply be found by constructing a vector of the worst objective function values. Then, the hypervolume can be calculated as the union of all the hypercubes, as shown in Equation 3.5. Figure 3.5 shows the union of all the hypercubes computed for the HV metric for a bi-objective minimization problem.

$$HV(PF) = volume\left(\bigcup_{i \in PF} v_i\right) \tag{3.4}$$

Considering the former definition, the relative hypervolume can be defined as:

$$RHV = \frac{HV(PF)}{HV(PF^*)} \tag{3.5}$$

Larger values of RHV indicate a more efficient and better distributed set of non-dominated solutions in the calculated Pareto front.

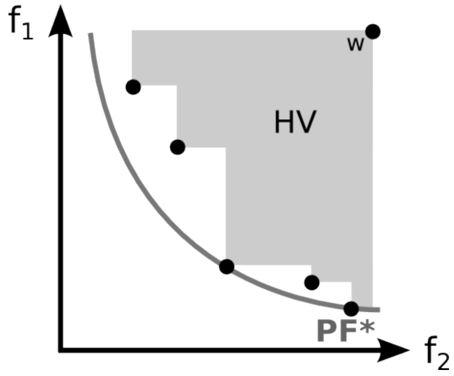


Figure 3.5: Union of all the hypercubes computed for the Hypervolume metric.

3.4.2 An example of a multi-objective metaheuristic solver: the PAES algorithm

Pareto archived evolution strategy (PAES) is a metaheuristic for solving multi-objective optimization problems (Knowles and Corne, 2000). Their creators argue that PAES may represent the simplest possible nontrivial algorithm capable of generating diverse solution for a MOP. Different variants of PAES have been proposed, in its simplest form, PAES is a trajectory-based local search metaheuristic which makes use of an external archive of solutions in order to store all the non-dominated solutions found during its execution. The simplest version of the PAES algorithm is outlined in Algorithm 4.

Algorithm 4 Pseudo-code of the PAES algorithm

```
1: s ← GenerateInitialSolution()
2: A = {s}
3: while stopping condition is not met do
4:   s' ← Mutate(s)
5:   if s dominates s' then
6:     /* discard s' */
7:   else if s' dominates s then
8:     AddToArchive(A,s')
9:     s ← s'
10:  else if s' is dominated by a ∈ A then
11:    /* discard s' */
12:  else
13:    TestAddToArchive(A,s')
14:    s ← SelectWorkingSolution(A)
15:  end if
16: end while
```

The PAES algorithm is comprised of three parts: the candidate solution generator, the solution acceptance function, and the archive of non-dominated solutions. The algorithm starts by generating an initial solution (line 1), it sets the initial solution as the working solution, and inserts it into the archive A (line 2). Then the algorithm iterates while the stopping criterion is not met. Each iteration the algorithm first performs a mutation of s and produces the candidate solution s' (line 4).

The *Mutate(s)* function is a simple random mutation that perturbs s randomly in order to generate s'. The candidate solution s' is then tested according to the acceptance function as follows: *i)* the candidate solution s' is discarded if it is dominated by the working solution s or by any solution in the archive A (line 6 and 11); *ii)* on the contrary, if s' dominates s, the working solution s is replaced by the candidate solution s' and s' is also inserted into the archive A (lines 8–9); *iii)* finally, if the candidate solution s' neither dominates nor is dominated by any solution $s^* \in \{s\} \cup A$ (i.e. all the solutions are equally good), then the candidate solution is tested to be added to the archive A and a new working solution is selected from the archive (lines 13–14).

A crucial part of PAES is the archive of non-dominated solutions. Because of performance issues, the archive of non-dominated solutions is size-bounded so only a limited number of solutions can be stored in the archive. It is not a trivial task to decide which solutions to store in the archive when the number of non-dominated solutions exceeds the maximum size of the archive. This crucial part is taken into account in function *TestAddToArchive(A,s′)* in line 13. Suppose the archive is at its full capacity and a new non-dominated solution s' is found, then the function *TestAddToArchive(A,s′)* can yield to discard the solution s' and not to include it in the archive A (even though is s' is non-dominated), or to discard some non-dominated solution currently in A in order to make room for solution s'. To this purpose a new crowding procedure called *adaptive grid algorithm* (AGA) was introduced as part of the PAES algorithm (Knowles and Corne, 2000). AGA is a novel archiving algorithm which guarantees to maintain the best solutions for each individual objective function in a MOP while at the same time storing a diverse set of non-dominated solutions. AGA will be thoughtfully described in Section 5.4, since it has been adopted into the local search methods proposed in this work for tackling the ME-HCSP.

3.5 Summary

This chapter presented a general notion on metaheuristics, its characteristics, and some classification criteria. The chapter also introduced the basis of stochastic optimization and presented a framework for *intelligently flexible* stochastic methods. It showed the relevance of local search methods in the exploitation phase of a metaheuristic and introduced the extended complexity of multi-objective optimization problems. It showed the applicability of metaheuristic methods on intractable combinatorial optimization problems, and the benefits of population-based metaheuristics when tackling multi-objective optimization problems. Finally, the ILS and PAES metaheuristics were described as examples of simple stochastic optimization methods. ILS is one of the most well-known stochastic local search based metaheuristic, and PAES is arguably the simplest nontrivial multi-objective metaheuristic which can compute accurate results for a MOP.

Next chapter will present a review of the most recent works that tackle the energy-aware scheduling problem in HC system.

Chapter 4

Related work

This chapter presents a review of the most recent works that have tackled variants of the energy-aware scheduling problem in HC environments. The chapter is organized as follows. In Section 4.1, those works tackling single-objective variants of the ME-HCSP are reviewed. In these works, the approaches usually consider energy consumption either as the objective function or as a problem constraint. The works reviewed in Section 4.2 formulate the ME-HCSP variants as multi-objective problems but use a single-objective approach in order to tackle them. These approaches convert the problem into a single-objective problem by using techniques such as aggregation of the objective functions or objective function prioritization. Finally, works reviewed in Section 4.3 formulate and tackle the ME-HCSP variants as true multi-objective problems.

4.1 Single-objective energy-aware scheduling

This section will present a review of some of the most recent works tackled as a single-objective variant of the ME-HCSP. In most of these works, the total energy consumption is considered as the unique problem objective function and some constraints are considered in order to guarantee a minimum system performance. The most recent single-objective related works are reviewed next.

Kim et al. (2007) proposed two online power-aware scheduling algorithms for scheduling a set of independent tasks (also known as *bag-of-tasks*) in order to minimize the total energy consumption. The problem formulation considers deadline constraints for the tasks, and makes use of a power management strategy using *Dynamic Voltage and Frequency Scaling* (DVFS) techniques. The proposed algorithm is based on the *Earliest Deadline First* (EDF) list-scheduling algorithm previously proposed by Dertouzos (1974). The experimental analysis was performed tackling dimensions of up to 16000 tasks and 32 machines, using machine scenarios and task workloads synthetically generated. Both algorithms were able to reduce energy consumption with little degradation of missing deadlines when comparing with the EDF.

Zhang et al. (2010) studied the offline scheduling problem considering a scenario comprised of a bag-of-tasks. In this, work six two-phase heuristics were proposed in order to minimize the total energy consumption while satisfying tasks deadlines constraints. The algorithm relies on a DVFS-enabled environment in order to manage the energy consumption. The experimental analysis is performed by tackling scenarios with dimensions up to 1000 tasks and 20 machines, and the computed results of the six proposed heuristics are compared with each other. Unfortunately, the proposed heuristics were not compared against any well-known heuristic.

Rizvandi et al. (2010) studied the problem of offline scheduling precedence-constrained tasks in order to minimize the energy consumption subject to tasks deadlines. The energy management in this work is achieved using a DVFS approach. The algorithm proposed by Rizvandi et al. (2010) is a heuristic algorithm based on the SRDVFS algorithm previously proposed by Kimura et al. (2006). Two sets of task graphs were tackled: randomly generated and real-world applications. Two real-world applications were used in the experiments, the LU decomposition and the Gauss-Jordan applications (Simunic et al., 2001). Scenarios with dimensions up to 500 tasks and 32 machines were generated using these task graphs, and the results were compared with the ones computed by the SRDVFS algorithm and by three different list-scheduling algorithms. The experiments showed the improved accuracy of the proposed algorithm.

Five offline scheduling algorithms for the minimization of the energy consumption considering makespan and tasks deadlines constraints were proposed by Apodaca et al. (2011). All of the proposed algorithms make use of DVFS mechanism for power management. Three of the proposed algorithms are metaheuristic-based: a Tree Search heuristic, based on the works by Wu et al. (2000) and Upadhyaya and Lata (2008); a Genetic Algorithm (GA), based on the works by Hartmann (2002) and Rahmani and Vahedi (2008); and a Tabu Search heuristic, based on the works by Zbigniew and Fogel (2000), Braun et al. (2001), and Lam et al. (2008). The remaining of the proposed algorithms are two two-phase list-scheduling heuristics: a MinMin-based heuristic and MinMax-based heuristic. These list-scheduling heuristics are used for comparison and to initialize the previously mentioned metaheuristic-based algorithms. Also a lower bound for the energy consumption objective is provided in order to compare the performance of the proposed algorithms. For the experimental analysis, scenarios were synthetically generated considering dimensions up to 4000 tasks and 25 machines. The results of the experimental analysis show the GA-based heuristic as the most accurate approach, achieving the lowest expected energy consumption while also meeting the problem constraints.

Zhu et al. (2011) proposed a novel scheduling strategy named *energy-efficient elastic* (3E) for online energy-aware scheduling a bag-of-tasks in a DVFS-enabled scenario. The proposed 3E strategy aims at minimizing the energy consumption with makespan as a secondary objective and considering tasks deadlines constraints. For the experimental analysis, a set of scenarios with dimension of 512 tasks and 128 machines were generated and three list-scheduling algorithms were proposed for comparison. The experimental results show that the 3E strategy outperforms the three list-scheduling algorithms proposed for comparison.

Ma et al. (2012) proposed an offline energy-aware list-scheduling approximation algorithm considering deadline constraints. The problem objective consists in minimizing the total energy consumption tackling a bag-of-tasks scenario. The energy management in this work follows an idle machine approach, considering a lower energy consumption when a machine is not computing a task. A lower bound for the problem is computed using *Integer Linear Programming* (ILP), and results are also compared with the EDF algorithm. The hardware specification parameters from 20 real-world servers were used for generating the machine scenarios. Scenarios with dimensions up to 100 tasks and 20 machines were generated in order to validate the algorithm. An online scheduling extension of the proposed algorithm is discussed and some initial experiments are performed using a log trace from the *Argonne National Laboratory* (ANL) Intrepid system, which was obtained from the *Parallel Workload Archive* (PWA). The ANL Intrepid system comprises a total of 163,840 computing cores with a theoretical computing performance of 557 TFlops, and the traces used contain accounting records from January 2009 to September 2009. Experimental results show that the offline version of proposed algorithm consumes 5% more energy than the computed lower bound, while the online version of the proposed algorithm improves the results computed by the EDF algorithm.

The work by Young et al. (2012) studied the problem of online scheduling a bag-of-tasks applications in an HC system in order to maximize the number of accepted tasks considering tasks deadlines and total energy consumption as constraints. A set of five heuristics using DVFS techniques were proposed in this work in order to tackle the problem. Two of the heuristics include a filtering mechanism to limit the set of feasible assignments the heuristic may use. For the experimental analysis, a set of scenarios considering dimensions of up to 1000 tasks and 8 machines where synthetically generated. Simulation results show that appropriate filtering mechanisms compute adequate results and are able to improve the performance heuristic algorithms performance.

In this section, a review of some of the most recent works tackling a single-objective problem variant of the ME-HCSP was presented. As previously stated, most of these works consider the total energy consumption as the unique problem objective function. In order to guarantee a minimum system performance threshold, these approaches consider some constraints such as maximum makespan, tasks deadlines, etc. Regarding the energy-aware approach, usually a DVFS technique is used in these works, although the work by Ma et al. (2012) which consider a low energy consumption idle state approach. Also, both online and offline schedulers were equally proposed for tackling the ME-HCSP variants in the reviewed works. As for the dimension of the scenarios tackled in these works, when considering the online scheduler approach, the higher dimension tackled was comprised of 16000 tasks and 32 machines; and when considering the offline scheduler approach, the higher dimension tackled was comprised of 4000 tasks and 25 machines. Most of the reviewed works designed simple heuristic methods for solving their proposed problem variants, such as list-scheduling heuristics. The only exception in the reviewed literature is the work by Apodaca et al. (2011) which designed three metaheuristics and two list-scheduling heuristics. In this work the proposed GA metaheuristic outperformed both of the proposed list-scheduling heuristics; but although the GA proved to be more accurate, it also required a considerably longer execution time than the list-scheduling heuristics.

4.2 Multi-objective energy-aware scheduling using a single-objective approach

The works reviewed in this section formulate the ME-HCSP variants as multi-objective problems but tackle the problem using a single-objective approach. Most of the reviewed literature used this approach, this is because the ME-HCSP is naturally a multi-objective optimization problem but single-objective optimization problems are significantly simpler to tackle. In order to tackle multi-objective problems as single-objective problem, techniques such as objective functions aggregation or objective functions prioritization, are used. The objective functions prioritization technique simply prioritizes one objective function over the others, and aims at optimizing the most important objective function, leaving the other objective functions as a secondary optimization goal. On the other hand, the objective functions aggregation technique optimizes an auxiliary objective function usually defined as the weighted sum of the objective functions. The most recent works which tackle multi-objective ME-HCSP variants using a single-objective approach, are reviewed next.

Kim et al. (2008) introduced several online multi-objective list-scheduling heuristics for scheduling tasks with deadlines and priorities, considering high-, medium-, and low-priority tasks. The problem is solved for an ad-hoc grid environment with limited battery capacity, in which DVFS techniques are used for power management. The problem does not follow a true multi-objective approach. Instead, the problem objectives are prioritized, being the primary goal to complete as many high-priority tasks by their deadlines as possible, and the secondary goal to maximize the sum of the weighted priorities of medium- and low-priority tasks completed by their deadlines. Seven list-scheduling heuristics are proposed in this work. For the experimental analysis, instances of dimension 50 tasks and 10 machines were used and the seven heuristics were compared with each other. The results of the experimental analysis show that the proposed algorithms based on the MinMin and Sufferage heuristics computed the most accurate results, but they required significantly longer execution time.

An offline two-phase scheduling heuristic was proposed by Lee and Zomaya (2009) for minimizing both makespan and energy consumption while considering precedence-constrained tasks. The energy management in this work is achieved using a DVFS approach. For the experimental analysis, both randomly generated tasks and real-world tasks were considered. Three real-world applications were used for the experimental analysis: the Laplace equation solver (Wu and Gajski, 1990), the LU-decomposition (Lord et al., 1983), and Fast Fourier Transformation (Cormen et al., 1990). Instances with dimensions up to 600 tasks and 64 machines were tackled and the results were compared with the previously proposed HEFT algorithm (Topcuoglu et al., 2002) and DBUS algorithm (Bozdag et al., 2006). The results of the experimental analysis showed the proposed algorithm is able to compute makespan metrics at least as accurate as HEFT and DBUS, and clearly outperforms both HEFT and DBUS when considering energy consumption. In this regard, it is important to note that neither HEFT nor DBUS consider energy consumption as a optimization objective.

Li et al. (2009) proposed an online energy-aware scheduler based on the MinMin list-scheduling heuristic in order to minimize makespan and energy consumption, the latter considering a low energy consumption of machines when in idle state. Again, a true multi-objective approach was not followed in this work as the proposed scheduler computes only one compromise solution considering both problem objectives. The experimental analysis was performed using instances of dimensions up to 10000×128. The proposed heuristic matched MinMin in makespan accuracy, and computed an energy reduction of up to 47%, again comparing to MinMin.

Khan and Ahmad (2009) developed an offline energy-aware grid scheduler based on the concept of Nash Bargaining Solution (NBS) from cooperative game theory. The proposed heuristic aims to simultaneously minimize the energy consumption and the makespan metric, subject to the constraints of task deadlines and architectural requirements. The energy management in this work is achieved using a DVFS approach. Even though the problem is formulated as a multi-objective problem, the approach for tackling the problem is not multi-objective. The proposed NBS approach aims at finding the *Bargaining Point* which is defined as the most suitable Pareto optimal solution. The experimental analysis was performed using dimensions up to 4370 tasks and 16 machines. Compared to an extension of the MinMin heuristic, the proposed algorithm showed an improvement of up to 22% in the makespan objective, and a reduction of up to 24% in the energy consumption objective.

Shekar and Izadi (2010) studied the problem of offline scheduling precedence-constrained tasks in DVFS-enabled machines in order to minimize both makespan and energy consumption. The authors propose an algorithm which is an extension of the *Dynamic Level Scheduling* (DLS) list-scheduling algorithm previously proposed by Sih and Lee (1993). For the experimental analysis, scenarios of dimension up to 200×5 were randomly generated. The results of the experimental analysis show that the proposed algorithm outperforms the algorithm originally proposed by Sih and Lee (1993).

A novel *Adaptive Power-Aware Scheduling* (APAS) strategy was proposed by Zhu et al. (2010). The proposed strategy tackles the problem of online scheduling a bag-of-tasks in a DVFS-enabled environment in order to simultaneously minimize the energy consumption and maximize the number of accepted deadline-constrained tasks. The APAS strategy is a two-phase heuristic comprised of two algorithms: the APAS1 and the APAS2. The former algorithm is responsible for assigning computational resources to arriving tasks, while the latter algorithm dynamically adjusts the voltage of each machine in order to minimize the energy consumption. The experimental analysis was performed tackling scenarios with dimensions up to 2048 tasks and 128 machines. The computed results were compared with the ones computed by two greedy heuristics: SLVL and SHVL (Yu and Prasanna, 2002). The experiments show that the proposed algorithms have competitive results, specially when tackling higher dimension scenarios with high arrival rates.

Garg et al. (2011) proposed a set of five list-scheduling heuristics to tackle the problem of online scheduling a set of independent deadline-constrained tasks in order to minimize the energy consumption and maximize the profit. The profit objective considers the CPU computing time to be charged by the minute, thus maximizing the profit implies maximizing the infrastructure utilization rate. The energy consumption objective relies in DVFS mechanisms for energy management. The set of proposed heuristics is comprised of two heuristics for energy consumption minimization, two heuristics for profit maximization, and the remaining heuristic which tackles both objectives simultaneously. To estimate the performance of the proposed algorithms, lower and upper bounds for the energy consumption and the profit are presented. Workload traces from the *Lawrence Livermore National Laboratory* (LLNL) Thunder system were used for the experimental analysis, containing accounting information comprising January 2007 to June 2007. For the experimental analysis, a set of eight real-world computing centers were considered and the first week of the workload traces was used. The experiments show that the proposed algorithms were able to compute schedules with only a 1% deviation from the upper bound of profit. Also, when considering a DVFS-enabled environment, the experiments showed an average reduction in energy consumption of 33%.

An offline two-phase energy-aware heuristic for grid scheduling was proposed by Pinel et al. (2011). The heuristic has two phases, in the first phase it applies the MinMin heuristic in order to find schedules with good makespan, in the second phase it applies a local search algorithm to further exploit the results computed during the first phase. The approach considers an ETC model extended with energy consumption consideration.

Instances of dimensions up to 512 tasks and 16 machines where used for the experimental analysis. The proposed method was able to compute schedules with similar quality to those computed by a Cellular Evolutionary Algorithm previously proposed by Alba and Dorronsoro (2008) for the same problem, while also reducing the execution time and significantly improving the MinMin schedules.

Kołodziej et al. (2011) developed two genetic-based energy-aware scheduling algorithms using DVFS for energy reduction. Both offline and online scenarios were considered, each one of them comprised of a set of independent tasks, and tackling the bi-objective scheduling problem of minimizing the makespan metric and the average energy consumption. The proposed algorithms do not follow a true multi-objective approach, but consider makespan as being the primary objective and energy consumption as the secondary one. The approach considers a ETC-based model extended with energy consumption consideration. Each solution of the initial population is generated either using the MCT heuristic, the LJFR-SJFR heuristic, or a random initialization. Two different acceptance criterion where used, one for each of the proposed algorithm. The first algorithm uses an *elitist* acceptance criterion in which only the best solutions remain in the population. The second algorithm uses a *struggle* mechanism in which new solutions are accepted by replacing a part of the population by the most similar individuals, if the new solutions improve the old ones. In the experimental analysis both algorithms are compared tackling scenarios of dimensions up to 4096 tasks and 256 machines, and executing for at most 40 seconds time. Both approaches achieved a considerable reduction in energy consumption when comparing against the initialization heuristics.

Diaz et al. (2011) evaluated three online list-scheduling heuristics for the minimization of the makespan objective and the total energy consumption objective considering a bag-of-tasks scenario. As the previous related work, the proposed heuristics make use of an extended ETC model in order to consider energy consumption using DVFS mechanisms for energy consumption management. The experimental analysis was performed using instances of dimension 512 tasks and 16 machines. Experiments results show that the proposed heuristics match MinMin in accuracy but with reduced execution time.

The article by Lindberg et al. (2012) introduced a set of eight heuristics for offline task scheduling in order to minimize makespan and the total energy consumption objectives, considering deadline and memory constraints. The set of proposed heuristics is comprised of six greedy list-scheduling heuristics and two genetic-based algorithms, neither of which followed a true multi-objective approach. The approach considers a ETC model extended with energy consumption consideration, making use of a DVFS mechanism for energy management. The experimental analysis was performed tackling scenarios of dimensions up to 1000 tasks and 16 machines for the two genetic-based heuristics, and up to 100000 tasks and 16 machines for the remaining greedy heuristics. Experimental results showed that the MinMin-based and the MaxMin-based list-scheduling algorithms where able to compute the most accurate solutions. The proposed genetic algorithm required considerably longer execution times than the six greedy list-scheduling heuristics. When solving the 1000×16 dimension instances, the fastest of the proposed genetic algorithms requires in average 38.5 minutes of execution time. On the other hand, the slowest of the proposed list-scheduling heuristic is able to solve the 100000×16 dimension instances in 21.2 seconds.

Sharifi et al. (2013) recently proposed PASTA: a power-aware solution for scheduling precedence-constrained tasks on heterogeneous computing resources. This work tackles the problem of offline scheduling precedence-constrained tasks in order to optimize the dual objective of minimizing the makespan metric and reducing the total energy consumption. PASTA is a two-phase algorithm which follows a idle machine energy strategy, considering a lower energy consumption when a machine is in idle state. For the experimental analysis, scenarios of dimensions up to 100 tasks and 20 machines were generated using task graphs of real-world applications such as: Gaussian Elimination (GE), Fast Fourier Transformation (FFT), LIGO, Epigenomics, and Montage workflows (Bharathi et al., 2008). The computed results were compared with the ones computed by the HEFT algorithm. Experimental results showed that PASTA produced schedules with approximately 20% longer makespan but with approximately 60% less energy consumption than HEFT's schedules. As previously stated, the HEFT algorithm does not consider energy consumption as a optimization objective.

This section presented a review of recent works which tackled multi-objective ME-HCSP variants using a single-objective problem approach. The reviewed works use techniques such as objective functions aggregation or objective functions prioritization, in order to tackle multi-objective problems as single-objective problems. As in the previous section, the DVFS technique is the most used energy-aware approach, although there are also a few works which consider a low energy consumption idle state approach. Both online and offline schedulers were proposed in the reviewed works, tackling online scenarios of up to 10000 tasks and 128 machines; and offline scenarios of up to 4096 tasks and 256 machines. Most of the reviewed works designed simple heuristic algorithmic solutions; although, when compared to the previous section, a greater number complex algorithmic solutions are applied, such as GA-based and LS-based methods.

4.3 True multi-objective energy-aware scheduling

The works reviewed in this section formulate the ME-HCSP variants as multi-objective problems and also design true multi-objective algorithmic approaches which compute a complete set of Pareto solution. The most recent works considering this approach are briefly summarized below.

A cooperative approach to solve the offline energy-aware scheduling problem was introduced by Mezmaz et al. (2011). This work proposed a parallel bi-objective hybrid genetic algorithm, improved with the energy-aware heuristics previously proposed by Lee and Zomaya (2011). The proposed approach considers precedence-constrained tasks and aims to minimize the makespan and the energy consumption, the latter using DVFS techniques. It follows a true multi-objective approach, and computes a complete Pareto front for the problem and reporting the hypervolume metric. Instances of dimensions up to 120 tasks and 64 machines were considered during the experimental analysis, and improvements of up to 12% for the makespan objective and up to 47% for the energy consumption objective were reported when comparing the proposed GA-based algorithm with the heuristics proposed by Lee and Zomaya (2011).

The work by Pecero et al. (2011) applied an offline bi-objective GRASP-based scheduling algorithm to minimize both makespan and the total energy consumption, while considering precedence-constrained tasks and inter-task communication costs. The proposed method builds a feasible solution using a greedy evaluation function, and uses a post-processing bi-objective local search method to improve the quality of the computed solution and to generate a set of Pareto solutions following a true multi-objective approach. The experimental analysis was performed tackling scenarios with dimensions up to 88 tasks and 32 machines and comparing the results computed by the proposed algorithm against the ones computed by a HEFT-based list-scheduling algorithm. Improvements of up to 7.78% for the makespan objective and up to 10.47% for the total energy consumption objective were reported.

Kessaci et al. (2011) proposed an online multi-objective genetic algorithm (MO-GA) for scheduling a set of independent tasks in order to minimize the energy consumption, to minimize the carbon dioxide emissions (CO_2), and to maximize the profit. The problem is subject to deadline-constrained tasks and relies in a DVFS-enabled infrastructure for its energy management. The genetic algorithm makes use of an external archive of non-dominated solutions in order to maintain the Pareto set of solutions. The decision making procedure is included as an integral part of the proposed solution, hence the algorithm provides only one final solution based on the selection parameters provided by the user. The author also proposes a greedy list-scheduling heuristic for population initialization which is also used as a reference baseline. One or two elements of the initial population are initialized using the greedy list-scheduling heuristic, while the rest of the population is generated randomly. The experimental analysis was performed using realistic workload traces of the LLNL from the Thunder cluster comprising accounting information from January 2007 to June 2007, and from RIKEN Integrated Cluster of Clusters (RICC) comprising accounting information from June 2010 to August 2010. The tasks workload is executed in a infrastructure comprised by 8 clusters (each ranging from 250 up to 2600 CPU), using a scheduling cycle of 50 seconds. The experimental results show that, when compared with the greedy list-scheduling heuristic, the MO-GA algorithm is able to reduce the energy consumption by 29.4%, the CO_2 emission by 26.3%, and to increment the profit by 3.6%. The authors did not evaluate the intermediate Pareto set desirable properties, hence some aspects of the proposed MO-GA were not analyzed.

Friese et al. (2012) studied the trade-offs between minimizing makespan and minimizing energy consumption when tackling offline scheduling problems in HC systems. The work considers a set of independent tasks and a idle-machine energy-aware strategy, which assumes a machine consumes less energy when it is in idle state. An extended ETC model is used for the problem, with the aggregation of machine energy consumption. The well-known NSGA-II (Deb et al., 2002) metaheuristic is applied to solving the problem, and two heuristics are used for population initialization: *i)* a MinMin heuristic for makespan minimization, and *ii)* an MCT-based heuristic for energy consumption minimization. Experimental analysis was performed tackling scenarios with dimensions of 1000 tasks and 50 machines. The computed results show that NSGA-II is able to optimize the initial population and provide a well defined Pareto front. Again, as in the previous related work, the Pareto set desirable properties are not evaluated. Various figures are presented in this work, showing the solution computed by the MinMin heuristic, and several Pareto fronts computed by the NSGA-II metaheuristic. Although no numerical results are presented, the presented figures clearly show that the NSGA-II metaheuristic matches the accuracy of the MinMin heuristic when considering the makespan objective function, and is able to outperform the accuracy of the MinMin heuristic when considering the energy consumption objective function.

The true multi-objective approach, despite being the most formally correct approach, is the least used approach for tackling ME-HCSP in the reviewed literature. This is because true multi-objective approaches have significantly higher computing requirements than single-objective approaches. The higher computing requirements has a considerable impact on the dimension of the scenarios which can be successfully tackled; the highest dimension scenarios tackled in the reviewed literature using this approach were comprised of only 1000 tasks and 50 machines. As for the algorithmic approaches, all of the reviewed works in this section designed complex algorithmic solutions, such as GA-based, or GRASP-based algorithms. This contrasts with the previous sections, where simple algorithmic solutions were preferred.

4.4 Summary

The energy-aware scheduling problem in heterogeneous systems is clearly a multi-objective problem. In its simplest formulation the problem must consider the energy consumption of the infrastructure and some system performance metric (e.g. makespan, task acceptance ratio, etc.). Taking this into consideration, there are three popular approaches in the literature to tackle energy-aware scheduling problems in heterogeneous systems.

The first approach consists in formulating the problem as a single-objective problem considering the other objective as a constraint. That is, either to maximize the performance of the system subject to a maximum energy consumption budget, or to minimize energy subject to a minimum system performance threshold.

The second approach consists in formulating the problem as a multi-objective problem, but tackling the problem as a single-objective problem. This can be done either by aggregating the energy consumption and system performance objectives into one unique weighed objective function, or by prioritizing one of the objectives.

The final approach is to formulate and tackle the problem as a multi-objective problem, computing a complete set of Pareto of trade-off solutions for the problem.

Most of the reviewed energy-aware scheduling works propose a list-scheduling heuristic and either tackle a single-objective optimization problem, or a multi-objective problem using a single-objective approach. This is usually because metaheuristic algorithms tend to require greater computational cost than greedy list-scheduling heuristics. On the other hand, the true multi-objective approach has been significantly less tackled; and when it is tackled, population-based metaheuristic algorithms are the preferred applied methods. This is because the additional computing cost of population-based metaheuristic algorithms is outweighed by their ability to compute a complete and accurate Pareto set in just one algorithm execution.

Regarding the energy management techniques, the main trend in energy-aware schedulers proposed in the related literature is to apply energy management methods within the computing elements (Kołodziej et al., 2012). In the literature we can find two different strategies for energy management: the *active* strategy, and the *passive* strategy. In the *active* strategy, the energy management decision parameters are embedded into the algorithmic approach. The most popular of these approaches use DVFS levels in order to actively lower the energy consumption of each machine in the system.

The *passive* strategy relies on hardware embedded energy saving features in order to support a machine low-energy state when the machine is idle; a machine is automatically considered to be in idle state when no computing task is assigned to it. These hardware-embedded energy-saving features are proprietary technologies and their efficiency depends on the hardware specification, for example AMD provides the Optimized Power Management technology in the Opteron processor family (Conway and Hughes, 2007), Intel provides the SpeedStep technology in Xeon/i7 processor family (Anshumali et al., 2010), etc.

As a final summary, Table 4.1 presents a brief outline of the works reviewed in this chapter.

Table 4.1: Summary of the reviewed related works.

Problem optimization approach		**Author/s (year)**	**Energy control strategy**	**Algorithmic method**	**Max. instance dimension** (tasks×machines)
	Single-objective	Kim et al. (2007)	active	EDF-based	16000×32
		Zhang et al. (2010)	active	Two-phase heuristic	1000×20
		Rizvandi et al. (2010)	active	List-scheduling	500×32
		Apodaca et al. (2011)	active	GA, TS, Tree search	4000×25
		Zhu et al. (2011)	active	EDF-based	512×128
		Ma et al. (2012)	passive	List-scheduling	100×60
		Young et al. (2012)	active	List-scheduling	1000×8
	Multi-objective as single-objective	Kim et al. (2008)	active	MinMin-based	50×10
		Lee and Zomaya (2009)	active	Two-phase heuristic	600×64
		Li et al. (2009)	passive	MinMin-based	10000×128
		Khan and Ahmad (2009)	active	NBS	4370×16
		Shekar and Izadi (2010)	active	DLS-based	200×5
		Zhu et al. (2010)	active	Two-phase heuristic	2048×128
		Garg et al. (2011)	active	List-scheduling	119849×8
		Pinel et al. (2011)	passive	Local search	512×16
		Kołodziej et al. (2011)	active	GA	4096×256
		Diaz et al. (2011)	active	List-scheduling	512×16
		Lindberg et al. (2012)	active	List-scheduling	100000×16
		Sharifi et al. (2013)	passive	Two-phase heuristic	100×50
	Multi-objective	Mezmaz et al. (2011)	active	GA-based	120×64
		Pecero et al. (2011)	active	GRASP-based	88×32
		Kessaci et al. (2011)	active	MO-GA	119849×8
		Friese et al. (2012)	passive	NSGA-II	1000×50

The next chapter will present the main algorithmic contribution of this thesis: the ME-MLS algorithm; a novel local search based heuristics for tackling the ME-HCSP. The ME-MLS is a parallel population-based metaheuristic which uses a true multi-objective approach and computes a complete Pareto set of trade-off solutions. The energy-aware approach proposed in the ME-MLS algorithm follows a passive strategy, considering the maximum and minimum machine energy consumption provided by the hardware manufacturer specification. Following this approach, a machine with assigned workload is assumed to operate at its peak performance, consuming the maximum energy detailed in its hardware specification. Conversely, a machine with no workload is assumed to be in idle state with the embedded energy saving technology keeping the energy consumption at its minimum specified value.

Chapter 5

ME-MLS: a true multi-objective algorithm for the ME-HCSP

This chapter presents the main algorithmic contribution of this thesis, the ME-MLS algorithm. The ME-MLS algorithm is a parallel population-based local search metaheuristic for tackling the ME-HCSP using a true multi-objective approach. The proposed algorithm aims at computing an accurate and diverse set of Pareto solutions for the ME-HCSP, while at the same time requiring reduced execution time.

This chapter is organized as follows. First, the general schema of the ME-MLS is described in order to present a general overview of the algorithm. The in-memory problem solution encoding is presented next, along with the population initialization heuristic. After that, two archiving algorithms for the ME-MLS are detailed. The design of the proposed embedded local search for the ME-MLS algorithm is presented. Finally, some implementation details are stated.

5.1 Algorithm design

The general outline of the ME-MLS algorithm is strongly based on the PAES algorithm, which was previously presented in Section 3.4.2. As previously stated, the ME-MLS algorithm aims at computing an *accurate* and *diverse* set of Pareto solutions in *reduced execution time.*

In order to compute a set of Pareto solutions, the ME-MLS algorithm maintains a size-bounded population of non-dominated solutions (or *elite* solutions). To guarantee the elite population to be as *diverse* as possible, an efficient archiving algorithm is applied. The archiving algorithm guarantees a diverse set of solutions avoiding biasing the search toward any of the two objectives of the ME-HCSP.

The ME-MLS makes use of an embedded Pareto-based local search heuristic for computing *accurate* solutions, iteratively improving the solutions in the elite population. In order for the local search algorithm to consider both ME-HCSP objectives, two different optimization strategies were designed: the *makespan* optimization strategy, and the *energy* optimization strategy. The *makespan* strategy focuses on improving the makespan objective function, while the *energy* strategy focuses on improving the total energy consumption objective function. Both strategies are applied during the algorithm execution.

A shared-memory parallel implementation is proposed in the design of the ME-MLS algorithm in order to maintain a *reduced execution time.*

This parallel implementation allows the ME-MLS to simultaneously improve multiple solutions from the population, taking advantage of modern multi-core architectures. The algorithm makes use of a pool of threads following a peer model, where no thread performs a master role, reducing the algorithm synchronization requirements. Algorithm 5 presents the logic of each thread in the ME-MLS algorithm.

Algorithm 5 Pseudo-code of each thread in the ME-MLS algorithm

Require: S size-bounded population of elite solutions
1: $s_{init} \leftarrow$ GenerateInitialSolution()
2: $S \leftarrow S \cup \{s_{init}\}$
3: **SynchronizationBarrier()**
4: **while** stopping criterion is not satisfied **do**
5:   **LockPopulation()**
6:   $s \leftarrow$ Choose a random solution from S
7:   $s' \leftarrow$ Clone s solution
8:   **UnlockPopulation()**
9:   $g \leftarrow$ Choose a random strategy (*makespan* or *energy*)
10:   $i \leftarrow$ Random number of iterations ($1 \leq i \leq$ `THREAD_IT`)
11:   **repeat**
12:     **for** $0 \rightarrow i$ **do**
13:       $s' \leftarrow$ LocalSearch(g, s')
14:     **end for**
15:     **if** s' dominates s **then**
16:       **TryLockPopulation()**
17:       **if** population is locked **then**
18:         TestAddToArchive(S, s')
19:         **UnlockPopulation()**
20:         search ends $\leftarrow$ *true*
21:       **else**
22:         $i \leftarrow$ Random number ($1 \leq i \leq$ `THREAD_IT/REWORK_FACTOR`)
23:         search ends $\leftarrow$ *false*
24:       **end if**
25:     **else**
26:       search ends $\leftarrow$ *true* {discard s'}
27:     **end if**
28:   **until** search ends
29: **end while**

Each thread starts by generating an initial solution s_{init} using some fast non-deterministic heuristic algorithm (line 1). It is important for the initialization algorithm to be non-deterministic in order to generate a diverse set of solutions for the initial population. After generating an initial solution, each thread adds its generated solution into the population and synchronizes itself with all the other threads in the thread pool (lines 2–3). Once the initial solutions are generated, all the threads start the main loop of the algorithm.

Every iteration, each thread starts by locking the population in order to gain exclusive access to it. Once the population is locked, the thread locking the population randomly chooses and clones a solution from the population, $s \rightarrow s'$ (lines 5–8).

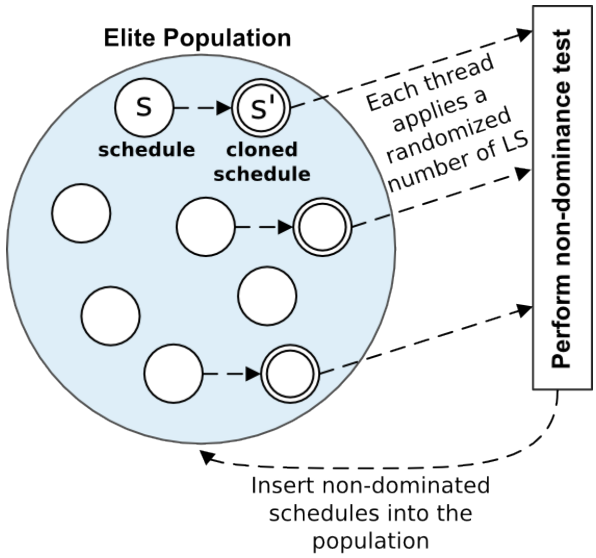


Figure 5.1: Diagram of the ME-MLS algorithm.

The locking thread unlocks the population and randomly chooses a search strategy g (either *makespan* or *energy*) and a random number of iterations i, bounded by the `THREAD_IT` parameter, for the local search (lines 8–10).

A strategy-dependent local search is applied i-times to the cloned solution s' hoping to improve it (lines 12–14). If the local search does not improve the solution s', i.e. the cloned solution s' does not dominate the original solution s, then the cloned solution s' is discarded and the current outer loop iteration finishes (line 26). Otherwise, if the solution s' dominates the solution s, then the function *TestAddToArchive()* is executed (line 18).

The function *TestAddToArchive()*, also known as *archiving algorithm*, must determine if the solution s' should be added to the size-bounded population S or not. An exclusive lock on the population must be issued in order to execute the *TestAddToArchive()* function, but a different locking approach is proposed this time because the *TestAddToArchive()* function can be quite computing intensive. This time each thread queries for a population lock, if the lock is granted, then the function *TestAddToArchive()* is executed. If the population is already locked by another thread, the threads which failed to lock the population will choose a reduced random number of additional local search iterations (line 22). These reduced additional local search iterations are bounded by the `THREAD_IT` parameter, and aim at providing further improvements to the solution s' while the population lock is not available. Once the additional local search iterations are performed, the thread queries for the population lock hoping this time the lock would be granted. If the population lock is still not granted, then another reduced number of additional local search iterations is chosen, and the inner loop of the search repeats itself until the lock is granted (lines 11–28). Figure 5.1 shows the general schema of the ME-MLS algorithm.

The ME-MLS algorithm uses a size-bounded Pareto-based approach for storing the solutions in the population, storing only non-dominated solutions. In population-based algorithms, the population must be size-bounded for the algorithm to efficiently compute an accurate solution set (Knowles and Corne, 2003).

The problem of dealing with size-bounded populations in multi-objective optimization algorithms has been throughly studied, and a number of archiving algorithms have been proposed for the implementation of the *TestAddToArchive()* function (Deb et al., 2002; Knowles and Corne, 2000; Zitzler et al., 2001). A pseudo-code of the generic Pareto-based *TestAddToArchive()* function is shown in Algorithm 6. The function starts by performing a non-dominance test between the candidate solution s' and every solution already in the population. If s' is dominated by any solution in the population, then s' is promptly discarded (line 2). If s' dominates one or more solution in the population, then the dominated solutions are discarded and s' is added to the population (lines 4–5). Finally, if s' do not dominates nor is dominated by any solution in the population, then all solutions are non-dominated and thus s' is added into the population. In this latter case, special care must be taken since the population is bounded in size. If the population exceeds its maximum capacity with the inclusion of the solution s', then one solution in S or s' itself must be discarded according to some criteria. In order to be effective, this criteria must address the diversity of the Pareto set, defined as one goal of the true multi-objective problem solving approach (lines 7–10).

Algorithm 6 Pseudo-code of the generic Pareto-based *TestAddToArchive()* function

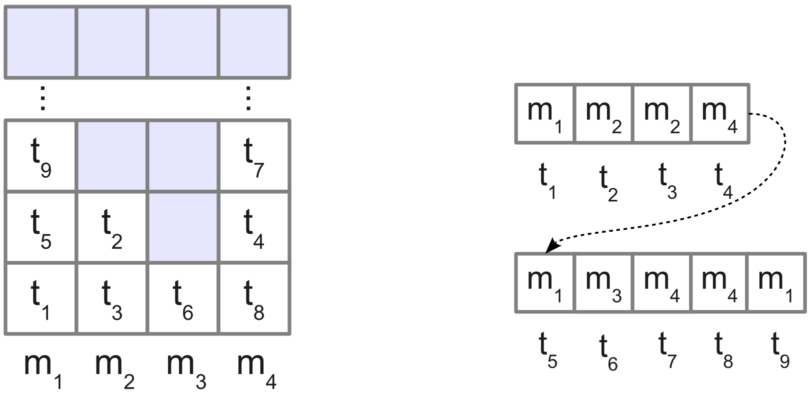
Require: S size-bounded population of solutions
    s' solution to be tested for its inclusion into S
1: **if** s' is dominated by any $s \in S$ **then**
2:     /* discard s' */
3: **else if** s' dominates any $s \in S$ **then**
4:     $S \leftarrow S \setminus \{\text{solutions dominated by } s'\}$
5:     $S \leftarrow S \cup \{s'\}$
6: **else**
7:     $S \leftarrow S \cup \{s'\}$
8:     **if** $|S| >$ `MAX_POPULATION_SIZE` **then**
9:         Discard the exceeding solution from S according to some criteria
10:     **end if**
11: **end if**

Two different versions of the ME-MLS algorithm were devised, depending on the archiving algorithm used: *i)* a fast greedy ad-hoc archiving algorithm and *ii)* the *Adaptive Grid Archiving* (AGA) algorithm proposed by Knowles and Corne (2000). Both archiving algorithms will be presented in detail in Section 5.4.

5.2 Problem encoding

Two well-known structures are proposed in the related literature for the in-memory encoding of the HCSP schedules, the *machine-oriented encoding* and the *task-oriented encoding* (Nesmachnow et al., 2010).

The *machine-oriented encoding* makes use of a bi-dimensional matrix representation to maintain one array of tasks for each machine. Each array contains the tasks assigned to be executed by the corresponding machine. For efficiency reasons the bi-dimensional structure is allocated statically in the ME-MLS algorithm, hence for encoding a schedule of up to N tasks and M machines, the bi-dimensional matrix should be $N \times M$ in size.



(a) Machine-oriented encoding (b) Task-oriented encoding

Figure 5.2: Encodings for in-memory representation of HCSP schedules.

The *task-oriented encoding* uses a uni-dimensional array representation. Each bucket in the array represents a task, and stores the value of the machine in which that task is to be executed. Again, for efficiency reasons the encoding structure is allocated statically in the ME-MLS algorithm, requiring an array of size N in order to encode schedules of up to N tasks.

Given a scenario of dimension 9×4, comprised by $P = \{m_1, m_2, m_3, m_4\}$ and $T = \{t_1, t_2, ..., t_9\}$. Suppose a schedule s is to be encoded in memory with the following tasks-to-machine assignments: $m_1 \leftarrow \{t_1, t_5, t_9\}$, $m_2 \leftarrow \{t_3, t_2\}$, $m_3 \leftarrow \{t_6\}$, and $m_4 \leftarrow \{t_8, t_4, t_7\}$. Figure 5.2a shows the schedule s encoded using a matrix machine-oriented encoding, and Figure 5.2b shows the schedule s encoded using a vector task-oriented encoding

ME-MLS maintains in memory a multi-structure comprising both a machine-based and a task-based encoding for schedules. This multi-structure allows ME-MLS to efficiently access the tasks assigned to a given machine in $O(1)$ via the machine-based encoding; and given a certain task, locate the machine to which is assigned also in $O(1)$ via the task-based encoding.

5.3 Population initialization

The initialization mechanism is a sensitive matter in the design of a metaheuristic algorithm. Providing a quality starting population, will most certainly improve the overall algorithm outcome. A good initialization mechanism should provide a diverse and accurate set of solutions. But because the ME-MLS aims at requiring reduced execution times, a good initialization mechanism for the ME-MLS should also be a fast mechanism.

Considering these properties, the rMCT list-scheduling heuristic, a randomized version of the MCT heuristic, is proposed for seeding the ME-MLS initial population.

The MCT is a simple heuristic which works by greedily minimizing the total computing time of the machine in which each task is scheduled. The scheduling is built one task at a time t_i, $i = 1...n$, assigning the task t_i to the machine m_j with the minimum completion time ct_{ij}. Algorithm 7 shows the pseudo-code of the MCT heuristic.

Algorithm 7 Pseudo-code of the MCT heuristic

```
Require: N number of tasks to be scheduled
    M number of available machines
for i = 1 → N do
    L ← ∅
    for j = 1 → M do
        ct_ij ← completion time of task t_i in machine m_j
        L ← L ∪ {ct_ij}
    end for
    Schedule task t_i into the machine index m_j with the minimum ct_ij ∈ L.
end for
```

The rMCT is a randomized version of the MCT. The rMCT heuristic randomly chooses a starting task t_s, a direction step (+1 or -1), and proceeds just as the MCT heuristic. Algorithm 8 shows the pseudo-code of the rMCT heuristic.

Algorithm 8 Pseudo-code of the rMCT heuristic

```
Require: N number of tasks to be scheduled
    M number of available machines
s ← Choose random starting task subindex
d ← Choose random starting direction step (d ∈ {+1, −1})
for i' = 1 → N do
    L ← ∅
    i ← (s + d × i')  mod N
    for j = 1 → M do
        ct_ij ← completion time of task t_i in machine m_j
        L ← L ∪ {ct_ij}
    end for
    Schedule task t_i into the machine m_j with the minimum ct_ij ∈ L
end for
```

The randomization provided by the rMCT heuristic, is used by the initialization mechanism to provide accurate and hopefully diverse starting schedules.

The MCT algorithm is not a among the most accurate list-scheduling heuristics (Braun et al., 2001), but is a very efficient heuristic. Both MCT and rMCT algorithms present a complexity order of $O(M \times N) \sim O(n^2)$, hence both are very fast algorithms and present very good scalability behavior.

5.4 Archiving algorithm

The elite population in ME-MLS is limited in size, so an archiving algorithm is required when the maximum size of the population is reached and the algorithm computes a new non-dominated solution. Two archiving methods were considered for the ME-MLS algorithm: a *Fast Greedy Ad-hoc Archiving* (FGAA) technique, and the *Adaptive Grid Archiving* (AGA) technique.

5.4.1 Fast Greedy Ad-hoc Archiving (FGAA)

This technique makes use of heuristic knowledge regarding the makespan and energy consumption metrics, and provides two basic archiving properties: *i)* it always inserts the newly found non-dominated schedules into the population, helping to prevent a stagnation situation; and *ii)* those schedules that compute the minimum value in at least one of the problem objective functions are never replaced. The algorithm works as follows. When the population is full, a schedule currently in the population is selected to be replaced based on a distance function, defined as the sum of the relative improvements of each objective metric.

The FGAA algorithm is a simple method that allows the local search procedure in ME-MLS to perform efficiently. On the other hand, the simple procedure in FGAA is not especially conceived to maintain high diversity in the population. When using the FGAA technique in the ME-MLS algorithm we will refer to ME-MLS$_{FGAA}$ algorithm.

5.4.2 Adaptive Grid Archiving (AGA)

The AGA technique was initially proposed as the archiving strategy for the PAES algorithm, proposed by Knowles and Corne (2000) and further formalized by Knowles and Corne (2003). The AGA algorithm works as follows. When the archive is not full, all non-dominated solutions are archived. When the archive is full, the algorithm tests which solution should be discarded based on how crowded together are the solutions in the archive. In order to compute the crowding of the solutions in the archive, the objective space is divided into hypercubes which define a multi-dimensional grid. In order to maintain diversity in the archive, the algorithm balances the density of non-dominated solutions in each one of the hypercubes. Each time a new solution s' is to be added, the grid location of s' in the solution space is determined. If the grid location of the new solution does not match with the most crowded hypercube, a solution belonging to that most crowded hypercube is removed and the new solution is inserted into the archive. However, to guarantee the archive maintains the Pareto front extreme solutions, all solutions which are extremal on any objective function are protected from removal. The multi-dimensional grid defined by the hypercubes must be constantly adapted in position and size, in order for the grid to cover all the archived solutions. Algorithm 9 presents the pseudo-code of the AGA algorithm.

The AGA algorithm starts by updating the boundaries of the adaptive grid (lines 1–4). After the grid is adapted to the current solutions in the archive, a dominance test is performed on the solution to be added to the archive. If the new solution s' is dominated by some existing solutions in the archive, then s' is discarded (lines 6–8). Otherwise, exactly one of the following cases can occur:

- If the new solution s' dominates some existing solutions in the archive, then all the dominated solutions are discarded and the new solution s' is added to the archive (lines 9–12).

- Otherwise, if the new solution s' is non-dominated with respect to S and the archive is full, then the new solution s' is added to the archive only if it extends the boundaries of the grid (lines 13–16), or if it is not located in the most crowded hypercube (lines 16–19). In both cases, a solution from the most crowded hypercube is removed in order to make room in the archive for the new solution s'.

- Last, if the new solution s' is non-dominated with respect to S and the archive is not at its full capacity, then the new solution s' is added to the archive (lines 22–24).

Algorithm 9 Pseudo-code of the AGA algorithm

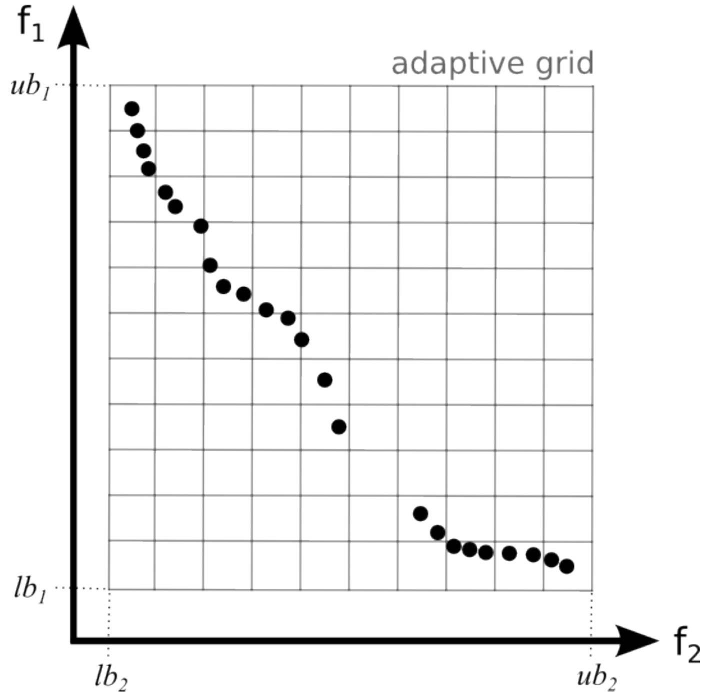
Require: K set of objectives in the MOP
  S archive of non-dominated solutions
  s' new non-dominated solution

1: **for all** $k \in K$ **do**
2:   $ub_k \leftarrow$ Compute the upper bound value of k for all $s \in S$
3:   $lb_k \leftarrow$ Compute the lower bound value of k for all $s \in S$
4:   Re-calculate grid boundaries for dimension k
5: **end for**
6: **if** s' is dominated by some $s \in S$ **then**
7:   /* discard s' */
8: **else**
9:   **if** s' dominates some $s \in S$ **then**
10:     Discard all $s \in S$ dominated by s'
11:     $S \leftarrow S \cup \{s'\}$
12:   **else if** $|S| =$ `MAX_ARCHIVE_SIZE` **then**
13:     **if** s is outside grid boundaries **then**
14:       Discard a solution from the most crowded hypercube
15:       $S \leftarrow S \cup \{s'\}$
16:     **else if** s is not in the most crowded hypercube **then**
17:       Discard a solution from the most crowded hypercube
18:       $S \leftarrow S \cup \{s'\}$
19:     **else**
20:       /* discard s' */
21:     **end if**
22:   **else**
23:     $S \leftarrow S \cup \{s'\}$
24:   **end if**
25: **end if**

Figure 5.3 shows an example of an AGA grid for a bi-objective minimization problem. It can be seen that some hypercubes are more crowded than others. If the archive is full, the solutions which fit in the most crowded hypercubes will be the first candidates to be discarded when new solutions are found.

The AGA strategy guarantees three very desirable properties for multi-objective optimization algorithms: *i)* it maintains solutions at the extremes of all objectives, *ii)* it maintains solutions in all of the Pareto occupied regions, and *iii)* it distributes the remaining solutions evenly among the Pareto regions. When using the AGA replacement technique in the ME-MLS algorithm we will refer to the ME-MLS$_{AGA}$ algorithm.

Figure 5.3: Example of an AGA grid for a bi-objective minimization problem.



5.5 Embedded Local search

The ME-MLS embedded local search is strongly based on the rPALS algorithm, which was originally proposed by Nesmachnow et al. (2012b) for tackling the HCSP. The rPALS algorithm, in turn, is based on the PALS algorithm by Alba and Luque (2007), which was proposed for the DNA fragment assembly problem. In order to fully describe the ME-MLS embedded local search, the already mentioned PALS and rPALS algorithms must be presented first.

5.5.1 The general schema of the PALS algorithm

The PALS algorithm (Alba and Luque, 2007) aims to be a very efficient local search heuristic for computing near-optimal solutions for large instances of hard to solve problems. It is based on the 2-opt heuristic proposed by Lin and Kernighan (1973), an heuristic originally proposed for solving the *traveling salesman problem* (TSP) (Korte and Vygen, 2007). Algorithm 10 shows the pseudo-code of the PALS algorithm proposed by Alba and Luque (2007).

In the PALS algorithm, $\mathcal{N}$ is the set of permutable elements of a solution. The algorithm starts by generating an initial working solution s using the *GenerateInitialSolution()* method (line 1). The algorithm evaluates the permutation of every tuple of elements (i, j), using the *CalculateDelta(s, i, j)* method, and maintains the evaluated permutations in the set L (lines 4–11). This δ-value represents a relative estimation measure of how much a given permutation of elements would improve a solution. It is worth noting that the algorithm only considers permutations which might improve the working solution ($\delta \geq 0$). Then, the algorithm selects a permutation from the set , and applies the selected permutation to the working solution s (lines 13–14).

Algorithm 10 Pseudo-code of the PALS algorithm

```
Require: 𝒩 is the set of permutable elements in a solution
s ← GenerateInitialSolution()
repeat
    L ← ∅
    for all i ∈ 𝒩 do
        for all j ∈ 𝒩/j ≠ i do
            δ ← CalculateDelta(s, i, j)
            if δ ≥ 0 then
                L ← L ∪ {(i, j, δ)}
            end if
        end for
    end for
    if L ≠ ∅ then
        (i, j, δ) ← Select best movement from L
        s ← Apply the movement defined by the tuple (i, j)
    end if
until no changes are applied to s
```

The algorithm iterates as long a it is able to find a permutation with a δ-value capable of improving s. The calculation of the δ-value of a permutation is a key issue in PALS. The δ-value is an estimate improvement measure and should not compute the problem objective function. Instead, a much more computationally efficient measure should be used.

5.5.2 rPALS algorithm for the HCSP

Nesmachnow et al. (2012b) proposed rPALS, a randomized variant of PALS for solving the HCSP using the ETC computing model. According to the HCSP formulation, we have a collection of tasks T that have to be scheduled on a collection of machines P. The goal in the HCSP is to minimize the total length of the schedule (or makespan). Refer to Section 2.3.1 for a complete formulation of the HCSP, and Section 2.3.2 for a formulation of the ETC computing model.

The rPALS algorithm improves the PALS algorithm for tackling the HCSP, two of the most remarkable improvements of rPALS are its stochastic nature and the multiple neighborhoods approach.

Regarding the former improvement, experiments showed that the original PALS algorithm does not scale efficiently when solving large HCSP scenarios. To tackle this problem, the deterministic approach in PALS was replaced by a stochastic approach in rPALS. Instead of evaluating all the possible permutations, rPALS only evaluates a random subset of the whole set of permutations. Because of the reduced size of the evaluated subset of permutations, rPALS is able to scale up to tackle real-world scenarios while maintaining reasonable execution times.

The second remarkable improvement in rPALS is the introduction of multiple search neighborhood structures. The rPALS algorithm uses two neighborhood structures simultaneously, the *swap* neighborhood structure and the *move* neighborhood structure.

The neighborhood structures for the rPALS algorithm are defined as follows:

- The *swap* neighborhood structure is defined by the swap of tasks operation, which swaps the assigned execution machines of a pair of tasks. For example, given two tasks t_1 and t_2 which are scheduled to be executed by machines m_1 and m_2 respectively. When the swap operator is applied, the tasks are swapped and task t_1 ends being scheduled to be executed in machine m_2 and task t_2 being executed by machine m_1.

- The *move* neighborhood structure is defined by the operation of moving a task from one machine to another. When the move operation is applied to the task t and the machine m, then the task t is re-assigned from its currently assigned execution machine to the machine m.

These two neighborhood structures are randomly applied during the search. The multiple search neighborhood structures increased the accuracy of the solutions computed by rPALS. Algorithm 11 presents the pseudo-code of the rPALS algorithm for the HCSP.

Algorithm 11 Pseudo-code of the rPALS algorithm for the HCSP

$s \leftarrow$ GenerateInitialSolution()
while stopping criterion is not satisfied **do**
$L \leftarrow \emptyset$
$m \leftarrow$ Choose a random machine
$n \leftarrow$ Choose a random neighborhood structure (*swap* or *move*)
if n is *swap* neighborhood **then**
while `RAND_MAX_TASKS` is not reached **do**
$t \leftarrow$ Choose a random task assigned to m in s
$m_{swap} \leftarrow$ Choose a random machine ($m \neq m_{swap}$)
$T_{swap} \leftarrow$ Select `RAND_MAX_TASK_SWAP` tasks from m_{swap} in s
for all $t_{swap} \in T_{swap}$ **do**
$\delta \leftarrow \text{CalculateDelta}_{swap}(s,\, t,\, t_{swap})$
if $\delta \geq 0$ **then**
$L \leftarrow L \cup \{(t, t_{swap}, \delta)\}$
end if
end for
end while
else if n is *move* neighborhood **then**
$m_{move} \leftarrow$ Choose a random machine ($m \neq m_{move}$)
$T_{move} \leftarrow$ Select `RAND_MAX_TASK_MOVE` tasks from m_{move} in s
for all $t_{move} \in T_{move}$ **do**
$\delta \leftarrow \text{CalculateDelta}_{move}(s,\, t_{move}, m)$
if $\delta \geq 0$ **then**
$L \leftarrow L \cup \{(t_{move}, m, \delta)\}$
end if
end for
end if
$s \leftarrow$ Apply the swap/move in L with the best δ-value
end while

The rPALS algorithm starts by generating an initial solution s using some list-scheduling algorithm (line 1). In the work by Nesmachnow et al. (2012b) the MCT algorithm is used for the initial solution generation. After generating the initial working solution, the algorithm enters its main loop until some stopping criterion is satisfied. Each iteration, the algorithm randomly selects a machine $m \in P$ and randomly selects a neighborhood structure $n \in \{swap, move\}$ (lines 3–4).

If the swap neighborhood structure is selected ($n = swap$), the algorithm starts an inner loop and iterates a number of times defined by the `RAND_MAX_TASKS` parameter. Each iteration, a task t is randomly chosen from the tasks assigned to be executed by the machine m in solution s. For each task t, a destination machine m_{swap} is randomly selected and a set of target tasks T_{swap} are also randomly selected from the tasks assigned to machine m_{swap} in s ($|T_{swap}| \leq$ `RAND_MAX_TASK_SWAP`). The δ-value is calculated for the swapping of each task t with every task $t_{swap} \in T_{swap}$. If the swap improves the working solution, then it is included in the set L of candidate movements (lines 7–17).

If the move neighborhood is selected ($n = move$), then a destination machine m_{move} is randomly selected and a set of tasks T_{move} are randomly selected from the tasks assigned to machine m_{move} in s ($|T_{move}| \leq$ `RAND_MAX_TASK_MOVE`). The δ value is calculated for the moving of every task in T_{move} into the machine m. If the move improves the working solution, then it is included in the set L (lines 19–25).

Last, the movement in L with the best δ-value is applied to the current solution s. Results showed that the rPALS algorithm is an accurate and efficient algorithm for tackling the HCSP.

5.5.3 ME-rPALS algorithm for the ME-HCSP

The ME-rPALS is a novel local search algorithm for tackling the ME-HCSP. It is strongly based on the rPALS algorithm and was specifically design to be used as the ME-MLS embedded local search.

Several improvements to the original rPALS were introduced in the ME-rPALS algorithm in order to tackle the ME-HCSP. The most notable modification in ME-rPALS with respect to the rPALS algorithm is the introduction of a second objective function: the total energy consumption objective function. The rPALS algorithm was designed for the HCSP, hence it considers only one objective function; on the other hand, the ME-rPALS must consider two objective functions in order to successfully tackle the ME-HCSP. For this purpose, two different optimization strategies were designed: the *makespan* optimization strategy, and the *energy* optimization strategy. The *makespan* strategy focuses on improving the makespan objective function, while the *energy* strategy focuses on improving the total energy consumption objective function. Algorithm 12 presents the pseudo-code of the ME-MLS embedded local search.

The local search algorithm receives the solution to be improved s and the improving search strategy g (*makespan* or *energy*) as input arguments. The ME-MLS algorithm starts by randomly selecting a machine m_{src}. In order to increase the accuracy of the algorithm, the method for randomly choosing the machine m_{src} does not follow a uniform distribution. Instead, if the selected optimization strategy g is the *makespan* strategy, then with high probability the machine m_{src} will be selected among the machines with the longest computing time; else if the selected optimization strategy g is the *energy* strategy, then with high probability the machine m_{src} will be selected among the machines consuming the most energy.

Algorithm 12 Pseudo-code of the ME-MLS local search algorithm

Require: s solution to be improved
  g search strategy applied to improve s
1: $L \leftarrow \emptyset$
2: $m_{src} \leftarrow$ Choose a random machine from s
3: $T_{src} \leftarrow$ Select `SRC_TASK_NHOOD` tasks from m_{src}
4: **for all** $t_{src} \in T_{src}$ **do**
5:   $n \leftarrow$ Choose a random neighborhood structure (*swap* or *move*)
6:   **if** n is *swap* neighborhood **then**
7:     $m_{dst} \leftarrow$ Choose a random machine from s ($m_{dst} \neq m_{src}$)
8:     $T_{dst} \leftarrow$ Select `DST_TASK_NHOOD` tasks from m_{dst}
9:     **for all** $t_{dst} \in T_{dst}$ **do**
10:       $(\delta_{makespan}, \delta_{energy}) \leftarrow \text{CalculateDelta}_{swap}(s,\, t_{src},\, t_{dst})$
11:       **if** $\delta_{makespan} \geq 0$ or $\delta_{energy} \geq 0$ **then**
12:         $L \leftarrow L \cup \{(t_{src}, t_{dst}, \delta_{makespan}, \delta_{energy})\}$
13:       **end if**
14:     **end for**
15:   **else if** n is *move* neighborhood **then**
16:     $M_{dst} \leftarrow$ Choose `DST_MACH_NHOOD` machines ($m_{src} \notin M_{dst}$)
17:     **for all** $m_{dst} \in M_{dst}$ **do**
18:       $(\delta_{makespan}, \delta_{energy}) \leftarrow \text{CalculateDelta}_{move}(s,\, t_{src},\, m_{dst})$
19:       **if** $\delta_{makespan} \geq 0$ or $\delta_{energy} \geq 0$ **then**
20:         $L \leftarrow L \cup \{(t_{src}, m_{dst}, \delta_{makespan}, \delta_{energy})\}$
21:       **end if**
22:     **end for**
23:   **end if**
24:   **if** g is *makespan* strategy **then**
25:     $s \leftarrow$ Apply swap/move in L with the best $\delta_{makespan}$ value
26:   **else if** g is *energy* strategy **then**
27:     $s \leftarrow$ Apply swap/move in L with the best δ_{energy} value
28:   **end if**
29: **end for**

After m_{src} is chosen, a set of tasks T_{src} from the ones assigned to be executed by machine m_{src} is selected (lines 2–3).

For each task $t_{src} \in T_{src}$, the algorithm randomly chooses a neighborhood structure n to perform the search (line 5). The ME-rPALS algorithm makes use of the same neighborhood structures as the rPALS algorithm, but choosing one neighborhood structure for each task t_{src} is a significant modification when comparing to the rPALS algorithm. In the rPALS algorithm only one neighborhood structure n is selected each outer iteration, hence the set L in rPALS may contain improvements found either using the *swap* or the *move* neighborhood structure, but not both. The ME-rPALS algorithm allows the set L to simultaneously have improvements found using the *swap* and the *move* neighborhood structures, hence allowing both neighborhoods to compete with each other during the search procedure. If the swap neighborhood structure is selected ($n = swap$), the algorithm randomly selects a machine m_{dst} and a set of tasks T_{dst} from the tasks assigned to m_{dst} (lines 7–8).

The δ-value tuple is calculated for the swapping of the previously selected task t_{src} and each task $t_{dst} \in T_{src}$ (lines 9–13). The algorithm only accepts into the set L the swapping of tasks which improve at least one of the objective functions (lines 11–13). Similarly, if the move neighborhood is selected ($n = move$), the algorithm randomly selects a set of destination machines M_{dst} (line 16). The δ-value tuple is calculated for the moving of the task t_{src} to each of the machines $m_{dst} \in M_{dst}$ (lines 17–22). Again, the algorithm only accepts moving of tasks which improve at least one of the objective functions (lines 19–21).

If the search strategy g is the makespan strategy, then the swap or move with the best $\delta_{makespan}$ is applied to the working solution s. Else, if g is the energy search strategy, then the swap or move with the best δ_{energy} is applied to s (lines 24–28).

The $CalculateDelta_{swap}(s, t_{src}, t_{dst})$ and $CalculateDelta_{move}(s, t_{src}, m_{dst})$ functions are a key part of the algorithm and provide a measure of how useful a given movement is. In order to fully define both $CalculateDelta()$ functions, a set of auxiliary functions must be defined first. Let $etc(t_i, m_j)$ be a function that, given a task t_i and a machine m_j, returns the time required by m_j to successfully execute t_i. Let $ct(m_j)$ be a function which, given a machine m_j, returns the time required by m_j to successfully execute all its assigned tasks (i.e. $\sum etc(t_i, m_j)$ for all t_i assigned to m_j). Let $e_{max}(m_j)$ be the amount of energy per time unit that machine m_j consumes when it is executing some task, and let $e_{idle}(m_j)$ be the amount of energy per time unit that machine m_j consumes when it is in idle state.

The $CalculateDelta_{swap}(s, t_{src}, t_{dst})$ computes the tuple $(\delta_{makespan}, \delta_{energy})$ as follows. Suppose t_{src} and t_{dst} are being swapped, let m_{src} be the machine to which t_{src} is currently assigned and let m_{dst} be the machine to which t_{dst} is currently assigned. Let CT_{src} be equal to $ct(m_{src})$, and CT_{dst} be equal to $ct(m_{dst})$. Let CT'_{src} be the compute time of the machine m_{src} after applying the given movement (i.e. $CT'_{src} = ct(m_{src}) - etc(t_{src}, m_{src}) + etc(t_{dst}, m_{src})$). And finally, let CT'_{dst} be the compute time of the machine m_{dst} after applying the given movement (i.e. $CT'_{dst} = ct(m_{dst}) - etc(t_{dst}, m_{dst}) + etc(t_{src}, m_{dst})$). The $\delta_{makespan}$-value for the $CalculateDelta_{swap}()$ is defined making use of the $CalculateDelta_{makespan}(CT_{src}, CT_{dst}, CT'_{src}, CT'_{dst})$ function which is defined in Equation 5.1. On the other hand, the δ_{energy}-value for the $CalculateDelta_{swap}()$ is defined making use of the $CalculateDelta_{energy}(CT_{src}, CT_{dst}, CT'_{src}, CT'_{dst})$ function which is defined in Equation 5.2

As for the $CalculateDelta_{move}(s, t_{src}, m_{dst})$ function. Suppose t_{src} is to be moved to m_{dst}, let m_{src} be the machine to which t_{src} is currently assigned. Again, let CT_{src} be equal to $ct(m_{src})$, and let CT_{dst} be equal to $ct(m_{dst})$. Let CT'_{src} be the compute time of the machine m_x after applying the movement, which this time is equal to $CT'_{src} = ct_{src} - etc(t_{src}, m_{src})$. Let CT'_{dst} be the compute time of the machine m_{dst} after applying the movement, $CT'_{dst} = ct_{dst} + etc(t_{src}, m_{dst})$. Again, using these new definitions, the $\delta_{makespan}$-value for the $CalculateDelta_{move}()$ is defined by the $CalculateDelta_{makespan}(CT_{src}, CT_{dst}, CT'_{src}, CT'_{dst})$ in Equation 5.1; and the δ_{energy}-value for the $CalculateDelta_{swap}()$ is defined by the $CalculateDelta_{energy}(CT_{src}, CT_{dst}, CT'_{src}, CT'_{dst})$ in Equation 5.2

$$\begin{aligned} &CalculateDelta_{makespan}(CT_x, CT_y, CT'_x, CT'_y) = \\ &\quad (CT'_x - max(CT_x, CT_y)) + (CT'_y - max(CT_x, CT_y) \end{aligned} \tag{5.1}$$

$$
\begin{aligned}
&CalculateDelta_{energy}(CT_x, CT_y, CT'_x, CT'_y) = \\
&\quad (CT_{src} - CT'_{src}) \times (e_{max}(src) - e_{idle}(src)) + \\
&\quad (CT_{dst} - CT'_{dst}) \times (e_{max}(dst) - e_{idle}(dst))
\end{aligned} \tag{5.2}
$$

5.6 Implementation details

ME-MLS is implemented in GNU C++ 4.6. Special care has been taken to avoid heavy C++ constructs like classes, interfaces, or polymorphism, in order to minimize the code execution overhead.

The multithreading support is provided by the GNU POSIX thread library 2.13. High level parallel multithreading libraries, such as OpenMP, were avoided in order to gain fine grain control over the synchronization mechanisms.

5.7 Summary

This chapter presented the ME-MLS algorithm, a true multi-objective method for computing a set of Pareto schedules which provide an accurate set of trade-off solutions for the ME-HCSP in reduced execution times. The ME-MLS algorithm heavily relies on a PALS-based embedded local search in order to compute accurate ME-HCSP schedules, while an archiving algorithm maintains a diverse set of Pareto solutions.

The next chapter will present the results computed by the experimental analysis performed on the ME-MLS algorithms. It will present a comparison between the ME-MLS$_{AGA}$ and ME-MLS$_{FGAA}$ archiving algorithms, as well as comparisons between different pseudo-random number generators, and an efficiency analysis of the algorithm.

Chapter 6

Experimental analysis

This chapter presents the comprehensive experimental analysis performed to evaluate the ME-MLS algorithm. The first sections presents the execution platform and the problem instances used in the experiments. After that, two additional techniques for solving the ME-HCSP are introduced in order to provide some results for baseline comparison. The first technique uses a linear programming relaxation approach to compute lower bound values for the problem, and the second technique uses list-scheduling heuristics to compute an accurate solution for the problem. A set of parameter settings experiments were performed for tunning the algorithm parameters, the next section presents the best parameters reported in these experiments. The following section presents the evaluation of three different pseudo-random number generators (PRNG) for improving the computational efficiency of the ME-MLS. In the main section, a comprehensive set of performed experiments are detailed in order to analyze: *i)* the quality of the solutions computed by the ME-MLS, and *ii)* the efficacy and diversity of the Pareto front computed by the ME-MLS computes. In addition, the final section presents a computational efficiency analysis of the ME-MLS algorithm.

6.1 Execution platform

The experimental analysis was performed on a 24-core Magny-Cours AMD Opteron Processor 6172, 2.1GHz, 24 GB RAM, running 64-bits CentOS 5.1 Linux, from the Cluster FING infrastructure (infrastructure web site: `http://www.fing.edu.uy/cluster`).

6.2 Problem instances

A set comprised of **792** ME-HCSP instances was generated to evaluate the ME-MLS algorithm. Each instance describes a machine *scenario* and a set of tasks *workload*. The machine *scenario* represents the underlying computing infrastructure, and the *workload* represents the work to be executed by the computing infrastructure. None of the computing models reviewed in the related literature represents adequately the ME-HCSP. The ETC model proposed by Ali et al. (2000) does not considers the infrastructure energy consumption, and the EMC model proposed by Nesmachnow et al. (2012a) considers a multi-core architecture infrastructure, which is not defined in the ME-HCSP as it is formulated in this thesis. Hence, the instances for the experimental analysis were generated using a hybrid ETC+EMC computing model.

The task workloads were synthetically generated using the methodology proposed by Ali et al. (2000), since it is a well-renowned methodology which was applied in various works in the related literature. On the other hand, the machine scenarios were generated using real-world data and following the methodology proposed by Nesmachnow et al. (2012a) in order to consider machine energy consumption in the problem scenario.

Three different instance dimensions were used in the experimental analysis (*number of tasks* $\times$ *number of machines*): 512×16, 1024×32, and 2048×64. For each dimension, 11 different scenarios and 24 tasks workloads were generated. Half of the tasks workloads were generated using the configuration parameters proposed by Ali et al. (2000), and the remaining half where generated using the configuration parameters proposed by Braun et al. (2001). Combining all these scenarios and workloads, a total of 264 ME-HCSP instances per dimension, that model realistic small- and medium-sized HC infrastructures were constructed for the experimental analysis.

6.3 Methods for baseline comparison

Two different approaches are proposed in order to provide a baseline reference for comparing the results computed by the ME-MLS algorithm. The first approach is based on linear programming relaxation techniques; for this approach two different linear programming formulations are constructed, one for each objective of the ME-HCSP. The second approach is based on a set of list-scheduling heuristic, four different MinMin-based heuristic are designed considering various combinations of the ME-HCSP objective functions.

6.3.1 Linear programming relaxation

Linear programming relaxation is a well-known technique for relaxing the integrality constraints of an integer programming NP-hard optimization problem with weaker non-integer conditions. Its goal is to transform the NP-hard problem into a linear programming problem, solvable in polynomial time (Knust and Brucker, 2006).

Consider an optimization problem where the objective function $f(x)$ is to be *minimized*. Suppose $\mathcal{S}$ to be the set of feasible solution of this problem, so that every feasible solution of the problem $x \in \mathcal{S}$. If a linear programming relaxation technique is applied to this problem, then the set of feasible solution for the original problem $\mathcal{S}$ is relaxed into a new set $\bar{\mathcal{S}}$, with $\mathcal{S} \subseteq \bar{\mathcal{S}}$.

Suppose $x^* \in \mathcal{S}$ is the optimal solution to the original problem, and $\bar{x^*} \in \bar{\mathcal{S}}$ is the optimal solution to the relaxed problem. The optimal solution for the relaxed problem $\bar{x^*}$ presents two important properties. First, $\bar{x^*}$ it is (most often) not a feasible solution for the original problem (i.e. $\bar{x^*} \notin \mathcal{S}$), since the integrality constraints of the original problem do not hold on the solutions for the relaxed problem. On the contrary, any solution for the original problem it is in fact a feasible solution for the relaxed problem, because $\mathcal{S} \subseteq \bar{\mathcal{S}}$. Second, the original problem is a sub-problem of the relaxed problem, hence the optimal solution $\bar{x^*} \in \bar{\mathcal{S}}$ solves the problem as least as optimally as the optimal solution $x^* \in \mathcal{S}$ (i.e. $f(\bar{x^*}) \leq f(x^*)$).

The quality gap between the optimal solution for the relaxed problem, and the optimal solution for the original problem, is known as the *integrality gap*, and it is defined as: $gap = f(x^*) - f(\bar{x^*})$.

The integrality gap is always greater or equal to zero, hence the objective function evaluation of the optimal solution for the relaxed problem can be considered a lower bound for the objective function evaluation of the original problem.

The ME-HCSP is formulated as a non-preemptive scheduling problem, hence the task-to-machine assignment decision variables are constrained to be integral. In the linear programming relaxation of the ME-HCSP, the integrality constraints on the task-to-machine assignment decision variables are relaxed, and the problem is transformed into a preemptive scheduling problem. In the resulting preemptive scheduling problem, any task can be interrupted during its execution, relocated to a different executing machine, and its execution can be resumed at a later time; all of this without considering any additional context switch execution costs.

Two separate linear programming relaxations are proposed for the ME-HCSP, one for each objective function. Algorithm 13 and 14 present the model formulation of the relaxed problems using the mathematical programming language AMPL (Fourer et al., 1990).

Algorithm 13 Linear programming relaxation model of the ME-HCSP for minimizing the makespan objective function

```
set TASK;
set MACHINE;
param ETC{t in TASK, m in MACHINE};
var x{t in TASK, m in MACHINE} ≥ 0;
var Makespan ≥ 0;
var MCT{m in MACHINE} ≥ 0;
minimize f : Makespan;
s.t. Task_is_assigned{t in TASK}: sum{m in MACHINE} x[t,m] == 1;
s.t. MCT_def{m in MACHINE}: MCT[m] = sum{t in TASK} x[t,m] * ETC[t,m];
s.t. Makespan_def{m in MACHINE}: Makespan ≥ MCT[m];
```

Algorithm 14 Linear programming relaxation model of the ME-HCSP for minimizing the total energy consumption objective function

```
set TASK;
set MACHINE;
param ETC{t in TASK, m in MACHINE};
param ENERGY_MAX{m in MACHINE};
param ENERGY_IDLE{m in MACHINE};
var x{t in TASK, m in MACHINE} ≥ 0;
var Makespan ≥ 0;
var MCT{m in MACHINE} ≥ 0;
minimize f :
    sum{t in TASK, m in MACHINE} x[t,m] * ETC[t,m] * ENERGY_MAX[m] +
    sum{m in MACHINE} (Makespan - MCT[m]) * ENERGY_IDLE[m];
s.t. Task_is_assigned{t in TASK}: sum{m in MACHINE} x[t,m] == 1;
s.t. MCT_def{m in MACHINE}: MCT[m] = sum{t in TASK} x[t,m] * ETC[t,m];
s.t. Makespan_def{m in MACHINE}: Makespan ≥ MCT[m];
```

Both models consider two sets of elements: the set of tasks ($TASK$) and the set of machines ($MACHINE$). Also, both models include the expected time to compute (ETC) parameter, a matrix with $TASK \times MACHINE$ elements, containing for each task its expected compute time value in each machine. Additionally, the model for minimizing the total energy consumption objective function includes the parameters $ENERGY\_MAX$ and $ENERGY\_IDLE$, which represent the energy consumption by time unit of each machine in each energy consumption state. Both models have the variables x, $Makespan$, and MCT. The x variable represent the task-to-machine assignment, and is the only true decision variable of the model, the remaining $Makespan$ and MCT variables are auxiliary variables for modeling the problem. The $Makespan$ variable represents the makespan metric value of the schedule, and the machine compute time (MCT) variable represents the local makespan of each machine. The objective function to be *minimized* in each model is defined exactly as it is defined in the ME-HCSP formulation. Finally, three constraints (s.t. or *subject to* sentences) are defined for each model. The $Task\_is\_assigned$ constraint is the only true constraint of the model, and it asserts that each task in $TASK$ is executed completely. The MCT_def and $Makespan\_def$ constraints are auxiliary constraints for asserting that the MCT and $Makespan$ auxiliary variables hold their true value.

The GNU Linear Programming Kit (GLPK) was used for solving the models presented in Algorithms 13 and 14 for each of the previously detailed generated problems instances. The revised simplex method was applied when solving the linear programming models for all of the problem instances.

The computed lower bounds (LB) for each objective function of the ME-HCSP, are useful to determine the accuracy of the results achieved using the ME-MLS algorithm. In order to measure the accuracy of the ME-MLS, the relative gap metric (*rgap*) is proposed. Given the objective function f, being $value_{ME-MLS}$ the value of f for a given solution computed by ME-MLS, and being $value_{LB}$ the optimal value of f for the relaxed problem. Then, the relative gap metric between the ME-MLS given solution and the LB is defined as shown in Equation 6.1.

$$rgap = \frac{value_{ME-MLS} - value_{LB}}{value_{LB}} \tag{6.1}$$

6.3.2 List-scheduling heuristics

In order to provide a feasible baseline reference for comparing the results computed by the ME-MLS algorithms, a set of list-scheduling heuristics based on the MinMin list-scheduling heuristic (Luo et al., 2007) were proposed.

MinMin heuristic for the HCSP

The MinMin list-scheduling heuristic is considered to be one of the most accurate heuristics for solving the HCSP, outperforming many other deterministic heuristics (Izakian et al., 2009b). The pseudo-code of the MinMin heuristic is presented in Algorithm 15.

The MinMin heuristic works by greedily picking the task that can be completed the soonest, taking into account the current machine assignments. The algorithm starts with a set U of all the *unmapped* tasks and a set of available machines P. For every task $t \in U$ it computes the completion time ct_{tm} of the task t when assigned to each machine $m \in P$.

Algorithm 15 Pseudo-code of the MinMin heuristic

Require: N number of tasks to be scheduled
M number of available machines
1: $U \leftarrow$ all tasks {set of unassigned tasks, $|U| = N$}
2: **while** $U \neq \emptyset$ **do**
3: $L \leftarrow \emptyset$
4: **for** each task $t_i \in U$ **do**
5: **for** $j = 1 \rightarrow M$ **do**
6: $ct_{ij} \leftarrow$ completion time of task t_i in machine m_j
7: $L \leftarrow L \cup \{ct_{ij}\}$
8: **end for**
9: **end for**
10: $ct^*_{ij} \leftarrow$ get the assignment with minimum completion time in L
11: Assign task t_i to machine m_j
12: Remove task t_i from L
13: **end while**

The task-to-machine assignment with the minimum overall completion time is applied, and the assigned task is removed from U. The process is repeated until all tasks are mapped and the set U is empty.

The MinMin heuristic presents three nested loops in its design, hence its execution complexity is in the order of $O\left(\frac{(N+1)\times N}{2} \times M\right) \sim O\left(\frac{N^2 \times M}{2}\right) \sim O\left(1/2 \times n^3\right) \sim O(n^3)$. Its $O(n^3)$ execution order makes the MinMin heuristic unusable for seeding the initial population. The MinMin heuristic is suitable for tackling small- to medium-sized scenarios, but may yield an excessive computing time when tackling large-sized scenarios.

MinMin-based heuristics for the ME-HCSP

MinMin uses a two-phase optimization strategy, depicted by the nested *for* cycle in lines 4–6 of Algorithm 15, where two different optimal assignments are selected (line 10 and 13). In the MinMin heuristic, both optimal assignments are chosen according to the same minimization criterion, the minimum completion time criterion. In order to consider the bi-objective scenario of the ME-HCSP, four MinMin versions were defined by alternating the minimization criterion in each phase.

The first version minimizes the completion time in both phases (i.e. the classic MinMin heuristic); the second version minimizes the energy consumption objective in both phases; in the remaining versions the minimization objectives are alternated to be in the first phase or the second phase. The *Min* notation is used when minimizing the completion time and the uppercase *MIN* notation when minimizing the energy consumption. Thus, the four versions of the MinMin heuristic are:

- *MinMin*: minimizes completion time in both phases.
- *MINMIN*: minimizes energy consumption in both phases.
- *MinMIN*: first minimizes completion time, and energy consumption second.
- *MINMin*: first minimizes energy consumption, and completion time second.

6.4 Parameter setting experiments

The goal of this work is to *efficiently* solve the ME-HCSP, thus a fixed 10 seconds execution time stopping criterion is used for the ME-MLS algorithm. This time stopping criterion is significantly lower than the execution time of metaheuristics in the related literature, which range from 40 seconds to 90 seconds (Kołodziej et al., 2011; Nesmachnow et al., 2012b). For the speedup evaluation of ME-MLS, the stopping criterion was set to 6 million iterations.

In order to provide statistical significance to the results and considering the stochastic nature of the proposed algorithm, 30 independent ME-MLS executions were performed on each instance. Each execution was performed using 24 threads, the maximum number of cores available in the computing platform.

A configuration analysis was performed using the 512×16 dimension instances in order to find the best values for the configuration of the ME-MLS algorithm parameters. Three sets of parameters of the ME-MLS algorithm were considered for the configuration analysis: the general schema parameters, the AGA parameters, and the ME-rPALS parameters. The general schema considered parameters are: the number of local search operations applied per iteration (`THREAD_IT`), and the re-work factor (`REWORK_FACTOR`) (see Algorithm 5). Only one parameter was considered for the AGA parameters set, the population size parameter (`MAX_ARCHIVE_SIZE`) (see Algorithm 9). The ME-rPALS considered parameters are used for defining the neighbourhood size in the local search: `SRC_TASK_NHOOD`, `DST_TASK_NHOOD`, and `DST_MACH_NHOOD` (see Algorithm 12).

The candidate values for the parameter settings study were: `MAX_ARCHIVE_SIZE` $\in \{30,34,38\}$, `THREAD_IT` $\in \{500,650,800\}$, `SRC_TASK_NHOOD` $\in \{24, 28, 32\}$, `DST_TASK_NHOOD` $\in \{16, 20, 24\}$, `DST_MACH_NHOOD` $\in \{8, 12, 16\}$, and `REWORK_FACTOR` $\in \{10, 14, 18\}$. The best results were obtained with the following configuration `POP_SIZE`=34, `THREAD_IT`=650, `SRC_TASK_NHOOD`=28, `DST_TASK_NHOOD`=16, `DST_MACH_NHOOD`=16, and `REWORK_FACTOR`=14.

6.5 Pseudo-random number generator analysis

In order to improve the computational efficiency of the proposed ME-MLS algorithm, three different *Pseudo-Random Number Generators* (PRNG) are analyzed. The PRNG is usually a significant time-consuming function in an iterative stochastic optimization method (Nesmachnow et al., 2011), hence improving the computational efficiency of the PRNG is a key issue for the ME-MLS to compute schedules in reduced execution time. A performance analysis was carried out in order to determine the relative contribution of the PRNG to the execution time of the ME-MLS algorithm when using the standard reentrant random function (`rand_r`) from the GNU standard C library. Figure 6.1 presents a the relative time contribution of the `rand_r` function after 5000 iterations. The reported results show that the `rand_r` function is the second most time consuming function of the ME-MLS algorithm, contributing with 18.2% of the total execution time. This results suggest that it is worth to study different PRNG to determine if significant reductions in the execution time can be obtained.

The performance of the ME-MLS was evaluated using three different PRNG: the reentrant random function (`rand_r`), and the reentrant 48-bit double-precision random function (`drand48_r`), both provided by the GNU standard C library; and the external Mersenne Twister (MT) generator (Matsumoto and Nishimura, 1998).

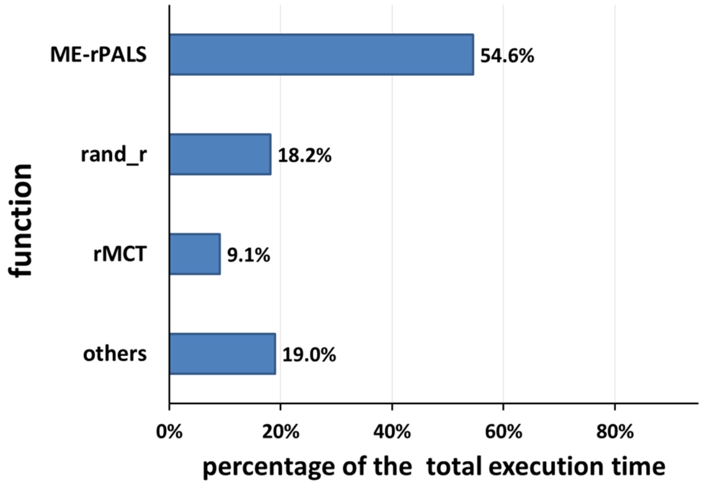


Figure 6.1: Performance analysis of the ME-MLS using the `rand_r` function.

Each method was studied by executing 15 independent executions with different number of parallel threads and using a stopping criterion of 6 million iterations. Table 6.1 reports the average and standard deviation values for the execution times versus the number of parallel threads, for each random number generation method.

Table 6.1: Execution times of the ME-MLS algorithm comparing different PRNG using different number of parallel threads.

number of threads	**execution time (s)**		
	`rand_r`	`drand48_r`	MT
1	31.5±0.24	38.2±0.24	**31.1±0.10**
2	35.2±1.68	65.0±3.26	**29.1±1.17**
4	39.4±1.59	26.0±1.09	**8.3±0.35**
6	18.0±0.79	31.5±2.12	**6.6±0.26**
8	19.1±0.47	15.4±0.70	**4.2±0.18**
10	19.6±0.58	14.7±0.71	**3.6±0.12**
12	19.5±0.67	9.4±0.41	**2.8±0.12**
14	19.6±0.53	10.3±0.34	**2.6±0.06**
16	20.1±0.82	7.4±0.39	**2.1±0.08**
18	20.2±0.87	7.2±0.22	**2.0±0.09**
20	19.7±0.57	5.7±0.18	**1.7±0.05**
22	12.2±0.50	5.7±0.14	**1.6±0.04**
24	12.3±0.36	4.6±0.10	**1.3±0.04**

The results in Table 6.1 show that the MT function is the best choice for significantly reducing the execution time of ME-MLS. Henceforth, all the experiments will be performed making use of the MT function for generating pseudo random numbers. Figure 6.2 graphically summarizes the results of the execution times of the ME-MLS for the three PRNG studied.

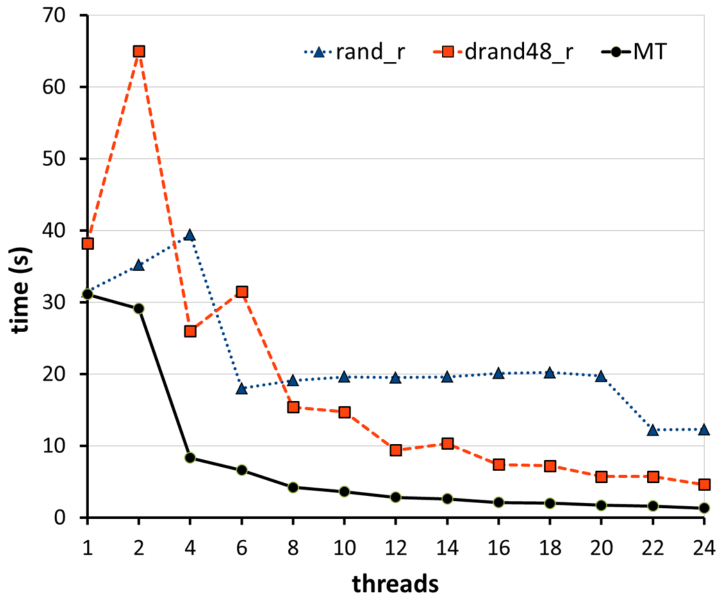


Figure 6.2: Average execution times of the ME-MLS algorithm comparing different PRNG using different number of parallel threads.

6.6 Results and discussion

This section presents and discusses the experimental results obtained in the evaluation of the ME-MLS scheduling algorithm.

This section starts by comparing the quality of the solutions computed by both of the ME-MLS algorithm variants, and the solutions computed by the proposed baseline reference comparison methods. In this first (most simple) comparison, the best computed solutions for each objective function of the ME-HCSP are compared separately, considering the objectives as being unrelated.

Then, a set of multi-objective optimization metrics are considered in order to evaluate the two variants of the proposed ME-MLS algorithm. Several evaluation metrics are considered for targeting both of the main goals of multi-objective optimization: *i)* the convergence of the computed solutions to the Pareto front, which depicts the efficacy or accuracy of the computed solutions; and *ii)* the correct sampling of the different trade-off solutions, which is represented by the diversity of the computed solutions (Deb, 2001).

6.6.1 Solution quality

The solution quality analysis compares the quality of the extremal solutions computed by the ME-MLS$_{AGA}$ and the ME-MLS$_{FGAA}$ algorithms, with the solutions computed by the proposed baseline reference methods.

The computed results for the proposed instances are reported grouping them by instance model, consistency type, and heterogeneity class. Each group consists of 11 different machine scenarios, each of which was independently solved 30 times. For each group, the *best* and average (*avg*) improvements are reported, for comparing the ME-MLS computed results with the MinMin-based heuristic which performed the best for the objective function and problem instance combination.

The average relative gap (*avg rgap*) is reported for each group, for comparing the ME-MLS computed results with the calculated lower bound for each instance.

In order to determine which ME-MLS algorithm performed better, a statistical analysis was performed over the improvement results computed by the ME-MLS$_{AGA}$ and the ME-MLS$_{FGAA}$ algorithms. First, the Kolmogorov-Smirnov (K-S) test was applied to check whether the metric values follow a normal distribution or not. The values for the D statistic by the K-S test indicated that the results for ME-MLS$_{AGA}$ and ME-MLS$_{FGAA}$ are not normally distributed. As a consequence, the non-parametric Kruskal-Wallis statistical test was performed with a confidence level of 95%, to compare the distributions of the improvements computed by ME-MLS$_{AGA}$ and ME-MLS$_{FGAA}$. Each result is emphasized in **bold** font when a given ME-MLS variant is always better than the other one in the 11 problem instances solved for each instance model, consistency type, and heterogeneity class.

Table 6.2 reports the *makespan* and *total energy consumption* for the 512×16 dimension instances. The results show that both ME-MLS variants outperform the best MinMin-based heuristic with makespan improvements of up to **24.4%** and total energy consumption improvements of up to **15.6%**. Both ME-MLS$_{AGA}$ and ME-MLS$_{FGAA}$ are able to compute similar schedules with no significant differences between them. The average relative gap for the makespan objective ranges from **1.6%** up to 10.5%, and a much tighter range of **2.3%** up to 6.7% for the total energy consumption objective.

Table 6.3 reports similar results for the 1024×32 dimension instances. Again, no ME-MLS variant outperforms the other in neither objective. The results show that both ME-MLS variants outperform the best MinMin-based heuristic with makespan improvements of up to **27.3%** and total energy consumption improvements of up to **16.1%**. The average relative gap for the makespan objective ranges from **3.1%** up to 11.9%, and a range of **2.4%** up to 8.0% for the total energy consumption objective.

Table 6.4 reports the computed results for the 2048×64 dimension instances. The computed results show that the ME-MLS$_{FGAA}$ algorithm computes slightly better schedules than ME-MLS$_{AGA}$ algorithm for a number of instances. Also, for a reduced number of instances, neither ME-MLS$_{AGA}$ nor ME-MLS$_{FGAA}$ were able to match the best MinMin-based heuristic accuracy with their computed the average makespan results. Despite this, both ME-MLS variants still outperform the best MinMin-based heuristic by a significant margin in the whole average of the 2048×64 dimension instances. The best MinMin-based heuristic is outperformed by both ME-MLS variants with makespan improvements of up to **29.1%** and total energy consumption improvements of up to **15.6%**. The average relative gap for the makespan objective ranges from **5.2%** up to 27.6%, and a range of **4.2%** up to 12.6%, for the total energy consumption objective.

The reported results show that, both ME-MLS algorithms, compute the best improvements on both objectives when solving the inconsistent type of instances, while considerably smaller improvements are computed when solving the consistent and semi-consistent type of instances. This demonstrate that the inconsistent type of instances are the harder to solve for the MinMin-based heuristics. Figure 6.3 summarizes the average improvements by dimension when comparing ME-MLS to each MinMin-based heuristic. It can be seen that the MinMin, the MinMIN and the MINMIN are very accurate and competitive heuristics, outperforming the MINMin heuristic by a large margin.

Table 6.2: ME-MLS makespan and energy consumption improvements over the best MinMin-based heuristic and lower bound relative quality gap for the 512×16 dimension instances.

	model	**consis-tency**	**hetero-geneity**	**ME-MLS$_{AGA}$** *best*	*avg*	*avg rgap*	**ME-MLS$_{FGAA}$** *best*	*avg*	*avg rgap*
makespan improvements	Ali	cons.	high high	12.9%	9.4%	3.6%	12.3%	9.1%	4.0%
			high low	14.0%	9.5%	3.7%	13.5%	9.2%	4.0%
			low high	10.9%	8.3%	2.9%	10.9%	8.4%	2.8%
			low low	10.9%	4.1%	10.4%	11.2%	4.0%	10.5%
		incons.	high high	21.4%	13.9%	4.1%	21.3%	13.9%	4.1%
			high low	24.4%	18.7%	4.6%	24.2%	18.7%	4.7%
			low high	23.5%	16.9%	3.6%	23.5%	16.9%	3.6%
			low low	18.5%	12.0%	6.7%	17.6%	11.8%	6.8%
		semi.	high high	15.9%	12.1%	4.7%	15.9%	11.9%	5.0%
			high low	15.5%	11.6%	5.3%	15.4%	11.3%	5.6%
			low high	16.9%	14.1%	3.5%	16.8%	14.1%	3.5%
			low low	12.8%	7.4%	10.2%	12.5%	7.3%	10.4%
	Braun	cons.	high high	11.7%	9.5%	3.5%	11.6%	9.2%	3.8%
			high low	5.5%	3.8%	2.2%	5.4%	3.8%	2.3%
			low high	10.3%	8.0%	4.6%	10.5%	7.8%	4.8%
			low low	5.9%	5.0%	1.6%	5.9%	5.0%	1.6%
		incons.	high high	23.5%	15.6%	3.9%	23.8%	15.5%	4.0%
			high low	11.8%	7.6%	3.0%	12.1%	7.6%	3.0%
			low high	17.8%	13.0%	3.9%	17.5%	12.9%	4.0%
			low low	7.6%	5.8%	2.4%	7.6%	5.9%	2.3%
		semi.	high high	18.8%	15.1%	3.7%	18.7%	14.9%	4.0%
			high low	7.7%	5.8%	2.3%	7.5%	5.8%	2.4%
			low high	16.9%	13.2%	3.7%	17.1%	13.2%	3.7%
			low low	10.1%	7.3%	1.9%	10.0%	7.4%	1.9%
	model	**consis-tency**	**hetero-geneity**	**ME-MLS$_{AGA}$** *best*	*avg*	*avg rgap*	**ME-MLS$_{FGAA}$** *best*	*avg*	*avg rgap*
energy consumption improvements	Ali	cons.	high high	9.5%	4.4%	5.5%	9.7%	4.4%	5.5%
			high low	9.5%	5.1%	6.3%	9.5%	5.2%	6.3%
			low high	9.3%	5.3%	3.9%	9.2%	5.2%	4.0%
			low low	10.7%	7.4%	3.1%	10.7%	7.4%	3.1%
		incons.	high high	12.6%	6.9%	4.0%	12.6%	6.9%	4.0%
			high low	15.0%	10.0%	4.6%	14.8%	10.0%	4.6%
			low high	15.5%	9.1%	3.5%	15.6%	9.2%	3.5%
			low low	12.2%	7.0%	4.3%	11.7%	7.0%	4.4%
		semi.	high high	8.6%	5.1%	5.1%	8.6%	4.9%	5.3%
			high low	7.9%	3.9%	6.4%	7.8%	3.7%	6.6%
			low high	10.2%	7.5%	3.5%	10.1%	7.5%	3.5%
			low low	8.5%	6.7%	3.7%	8.9%	6.7%	3.6%
	Braun	cons.	high high	9.7%	6.3%	5.5%	10.0%	6.2%	5.6%
			high low	5.3%	2.3%	3.1%	5.2%	2.3%	3.1%
			low high	8.3%	4.1%	6.7%	8.3%	4.2%	6.6%
			low low	5.5%	3.2%	2.6%	4.7%	3.1%	2.7%
		incons.	high high	13.4%	7.9%	3.8%	13.6%	7.9%	3.9%
			high low	7.0%	3.7%	3.0%	7.2%	3.7%	3.0%
			low high	10.3%	6.6%	3.8%	10.5%	6.5%	4.0%
			low low	4.3%	2.8%	2.4%	4.4%	2.8%	2.3%
		semi.	high high	12.8%	8.9%	4.3%	12.5%	8.8%	4.5%
			high low	6.4%	3.8%	2.7%	6.3%	3.7%	2.8%
			low high	11.1%	7.6%	4.4%	10.9%	7.7%	4.4%
			low low	7.2%	4.9%	2.4%	7.3%	4.9%	2.5%

Table 6.3: ME-MLS makespan and energy consumption improvements over the best MinMin-based heuristic and lower bound relative quality gap for the 1024×32 dimension instances.

	model	**consis-tency**	**hetero-geneity**	**ME-MLS$_{AGA}$** *best*	*avg*	*avg rgap*	**ME-MLS$_{FGAA}$** *best*	*avg*	*avg rgap*
makespan improvements	Ali	cons.	high high	9.7%	7.3%	4.4%	10.5%	7.1%	4.7%
			high low	8.7%	7.2%	5.0%	9.7%	6.7%	5.5%
			low high	10.8%	8.7%	3.7%	11.3%	9.0%	3.5%
			low low	6.8%	2.3%	9.2%	7.0%	2.6%	8.9%
		incons.	high high	23.9%	14.6%	6.2%	24.3%	14.6%	6.3%
			high low	24.3%	16.6%	6.4%	24.5%	16.6%	6.5%
			low high	26.9%	17.7%	5.2%	27.3%	17.6%	5.3%
			low low	25.4%	11.1%	11.7%	24.0%	11.0%	11.9%
		semi.	high high	14.4%	11.3%	6.9%	14.3%	11.1%	7.2%
			high low	13.1%	10.6%	7.1%	13.3%	10.4%	7.4%
			low high	15.0%	11.1%	6.9%	15.1%	11.2%	6.9%
			low low	10.5%	6.8%	11.3%	10.6%	6.6%	11.5%
	Braun	cons.	high high	8.8%	6.2%	4.4%	9.1%	6.1%	4.6%
			high low	10.1%	7.6%	4.6%	10.3%	7.5%	4.8%
			low high	8.5%	6.0%	4.6%	8.7%	5.8%	4.9%
			low low	10.5%	9.0%	3.5%	11.3%	9.3%	3.1%
		incons.	high high	23.6%	19.1%	6.0%	23.3%	19.0%	6.1%
			high low	22.3%	14.8%	6.2%	22.5%	14.7%	6.3%
			low high	22.9%	16.9%	6.3%	23.0%	16.8%	6.5%
			low low	22.3%	16.5%	5.0%	22.0%	16.5%	5.0%
		semi.	high high	16.7%	12.0%	7.8%	16.6%	11.7%	8.2%
			high low	14.7%	11.5%	7.5%	14.5%	11.4%	7.6%
			low high	15.6%	11.9%	7.8%	15.5%	11.6%	8.2%
			low low	14.9%	12.1%	5.9%	14.8%	12.3%	5.7%
	model	**consis-tency**	**hetero-geneity**	**ME-MLS$_{AGA}$** *best*	*avg*	*avg rgap*	**ME-MLS$_{FGAA}$** *best*	*avg*	*avg rgap*
energy consumption improvements	Ali	cons.	high high	9.6%	5.9%	6.0%	10.0%	6.2%	5.7%
			high low	8.9%	5.6%	7.1%	9.2%	5.4%	7.3%
			low high	9.9%	7.8%	4.0%	9.8%	7.3%	4.5%
			low low	11.4%	8.9%	2.4%	11.4%	8.8%	2.4%
		incons.	high high	13.9%	6.7%	6.1%	14.3%	6.7%	6.2%
			high low	13.5%	7.6%	6.3%	13.7%	7.6%	6.4%
			low high	15.8%	9.3%	5.0%	16.1%	9.2%	5.1%
			low low	15.8%	7.0%	6.7%	15.2%	7.0%	6.7%
		semi.	high high	6.9%	4.5%	6.8%	7.6%	4.3%	7.1%
			high low	6.5%	3.3%	6.7%	6.0%	3.2%	6.9%
			low high	8.2%	4.9%	5.6%	7.6%	4.6%	5.9%
			low low	7.7%	5.8%	3.9%	7.6%	5.8%	4.0%
	Braun	cons.	high high	9.3%	5.7%	6.1%	10.3%	6.3%	5.3%
			high low	9.4%	6.0%	6.6%	9.8%	6.3%	6.4%
			low high	7.9%	4.4%	6.1%	8.0%	4.7%	5.9%
			low low	11.3%	8.8%	3.9%	11.0%	8.5%	4.2%
		incons.	high high	13.8%	9.7%	5.9%	13.5%	9.6%	6.0%
			high low	12.0%	6.8%	6.1%	12.1%	6.7%	6.2%
			low high	13.1%	8.2%	6.2%	13.2%	8.0%	6.4%
			low low	12.4%	8.5%	4.8%	12.1%	8.5%	4.8%
		semi.	high high	9.1%	4.2%	7.3%	9.6%	3.8%	7.8%
			high low	7.0%	3.1%	7.1%	6.7%	2.8%	7.5%
			low high	6.3%	4.0%	7.7%	6.9%	3.7%	8.0%
			low low	9.4%	6.6%	4.3%	9.6%	6.4%	4.6%

Table 6.4: ME-MLS makespan and energy consumption improvements over the best MinMin-based heuristic and lower bound relative quality gap for the 2048×64 dimension instances.

	model	**consis-tency**	**hetero-geneity**	**ME-MLS$_{AGA}$**			**ME-MLS$_{FGAA}$**		
				best	*avg*	*avg rgap*	*best*	*avg*	*avg rgap*
makespan improvements	Ali	cons.	high high	7.2%	5.0%	6.9%	8.5%	5.7%	6.1%
			high low	6.2%	4.2%	6.6%	7.7%	5.2%	5.5%
			low high	6.8%	4.5%	7.8%	7.6%	6.3%	5.9%
			low low	3.4%	-1.0%	13.8%	4.8%	1.1%	11.4%
		incons.	high high	27.4%	21.8%	8.5%	27.3%	21.6%	8.7%
			high low	25.6%	18.8%	8.9%	25.5%	18.6%	9.2%
			low high	22.1%	15.8%	8.3%	21.7%	15.9%	8.2%
			low low	14.7%	3.7%	27.6%	14.3%	3.8%	27.3%
		semi.	high high	11.8%	9.2%	11.2%	13.6%	9.2%	11.2%
			high low	10.9%	8.5%	10.5%	13.6%	8.6%	10.4%
			low high	9.7%	7.5%	10.1%	10.1%	8.6%	8.7%
			low low	5.3%	-1.4%	24.0%	5.7%	-0.7%	23.2%
	Braun	cons.	high high	7.2%	4.7%	6.9%	8.1%	5.4%	6.0%
			high low	7.6%	6.2%	5.2%	7.8%	6.4%	4.9%
			low high	7.4%	5.0%	6.6%	8.8%	5.6%	5.9%
			low low	6.4%	4.3%	7.6%	8.0%	6.0%	5.7%
		incons.	high high	24.7%	15.5%	8.9%	24.6%	15.2%	9.3%
			high low	29.1%	21.2%	8.4%	28.9%	20.9%	8.7%
			low high	24.2%	18.2%	8.7%	23.7%	17.9%	9.1%
			low low	26.9%	19.4%	8.1%	26.9%	19.5%	7.9%
		semi.	high high	10.9%	8.3%	10.7%	12.8%	8.4%	10.7%
			high low	12.7%	9.9%	9.0%	13.0%	10.3%	8.4%
			low high	12.7%	10.1%	10.1%	13.9%	10.0%	10.1%
			low low	14.6%	10.8%	10.3%	15.5%	12.2%	8.7%

	model	**consis-tency**	**hetero-geneity**	**ME-MLS$_{AGA}$**			**ME-MLS$_{FGAA}$**		
				best	*avg*	*avg rgap*	*best*	*avg*	*avg rgap*
energy consumption improvements	Ali	cons.	high high	9.4%	4.6%	10.0%	11.1%	6.8%	7.5%
			high low	9.2%	4.6%	8.7%	10.7%	6.5%	6.4%
			low high	12.7%	9.9%	4.8%	12.7%	9.4%	5.3%
			low low	12.9%	9.3%	4.6%	13.3%	9.2%	4.7%
		incons.	high high	14.1%	10.2%	8.2%	14.8%	10.0%	8.4%
			high low	14.3%	8.6%	8.6%	14.3%	8.4%	8.9%
			low high	11.1%	7.6%	6.9%	11.0%	7.5%	7.0%
			low low	10.0%	4.1%	12.6%	10.0%	4.1%	12.5%
		semi.	high high	6.1%	2.1%	11.1%	7.8%	2.3%	10.9%
			high low	7.4%	2.8%	9.7%	7.3%	2.5%	10.0%
			low high	7.7%	5.8%	5.8%	7.3%	5.2%	6.5%
			low low	11.2%	9.2%	4.2%	11.2%	9.1%	4.3%
	Braun	cons.	high high	9.9%	5.3%	8.8%	10.8%	7.3%	6.5%
			high low	9.5%	6.6%	8.5%	10.4%	8.3%	6.5%
			low high	9.9%	4.8%	9.4%	11.7%	7.0%	6.9%
			low low	13.9%	10.1%	4.7%	13.8%	9.6%	5.3%
		incons.	high high	12.4%	6.1%	8.6%	12.1%	5.8%	9.0%
			high low	15.6%	9.8%	8.2%	15.5%	9.6%	8.5%
			low high	13.1%	7.8%	8.3%	12.4%	7.4%	8.8%
			low low	14.1%	9.6%	6.9%	14.4%	9.6%	6.9%
		semi.	high high	7.3%	2.6%	10.7%	8.2%	2.9%	10.4%
			high low	7.7%	4.5%	9.4%	8.8%	4.5%	9.4%
			low high	8.7%	3.6%	9.9%	9.8%	3.5%	9.9%
			low low	11.0%	8.0%	6.4%	11.1%	7.5%	7.0%

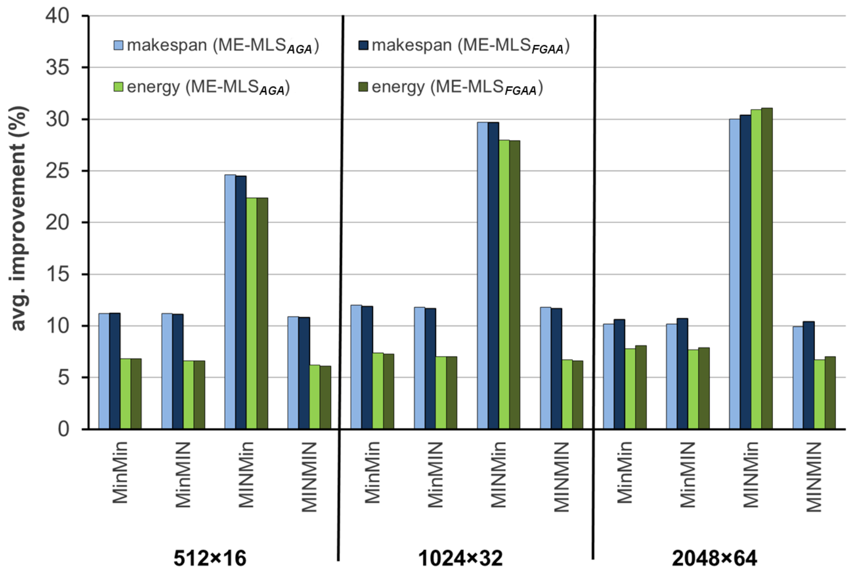


Figure 6.3: Average ME-MLS improvements over the MinMin-based heuristics.

Table 6.5: ME-MLS makespan improvements summary over the best MinMin-based heuristic.

dimension	ME-MLS$_{AGA}$			ME-MLS$_{FGAA}$		
	best	*avg*	*avg rgap*	*best*	*avg*	*avg rgap*
512×16	24.4%	10.3%	4.2%	24.2%	10.2%	4.3%
1024×32	26.9%	11.2%	6.4%	27.3%	11.1%	6.5%
2048×64	29.1%	9.6%	10.2%	28.9%	10.1%	9.6%

Tables 6.5 and 6.6 summarize the the total average relative gap for each dimension with respect to the calculated lower bound, and the total average and best improvements, again comparing each ME-MLS algorithm with the best MinMin-based heuristic. The results show the best average makespan objective improvements are computed when solving the 1024×32 dimension instances, while the average energy consumption objective improves as the instance dimension increases. The best improvement on the average energy consumption objective is computed when solving the 2048×64 dimension instances. Regarding the calculated lower bound, the results demostrate that the average *rgap* increases for both objectives as the instance dimension increases, showing there is potentially more room for improvements as the dimension of the instances increases.

Table 6.6: ME-MLS energy consumption improvements summary over the best MinMin-based heuristic.

dimension	ME-MLS$_{AGA}$			ME-MLS$_{FGAA}$		
	best	*avg*	*avg rgap*	*best*	*avg*	*avg rgap*
512×16	15.5%	5.9%	4.1%	15.6%	5.8%	4.2%
1024×32	15.8%	6.4%	5.8%	16.1%	6.3%	5.9%
2048×64	15.6%	6.6%	8.1%	15.5%	6.8%	7.8%

6.6.2 Multi-objective optimization metrics

In this section, the ME-MLS$_{AGA}$ and ME-MLS$_{FGAA}$ algorithms are compared with each other studying a set of multi-objective optimization metrics for the Pareto front approximations computed by each algorithm. Each multi-objective optimization metric used for the comparison can be classified either as a *efficacy* metric, a *diversity* metric, or an *hybrid* metric, depending on the multi-objective optimization goal it measures.

The *efficacy* metrics evaluate the convergence towards the Pareto front. We consider in this work two quality metrics: the number of (different) non-dominated solutions found for each algorithm (*ND*); and the Inverted Generational Distance (*IGD*), defined as the normalized sum of the distances between the non-dominated solutions in the Pareto front found by the algorithm and a set of uniformly distributed points in the true Pareto front. Smaller values of IGD mean a better approximation to the Pareto front. The reported IGD-value is normalized with respect to the best IGD-value computed between ME-MLS$_{AGA}$ and ME-MLS$_{FGAA}$, i.e. $IGD_{value} = \frac{IGD_{value}}{IGD_{best}}$.

The *diversity* metrics measure the distribution of the computed non-dominated solutions, evaluating the correct sampling of the target Pareto front. We consider in this work the *Spread* metric for evaluating the diversity of the computed results.. The spread metric includes information about the extreme points of the true Pareto front in order to compute the spacing between the points of the Pareto front. Smaller values of spread mean a better distribution of non-dominated solutions in the calculated Pareto front. As with the IGD metric, the reported Spread-value is normalized with respect to the best Spread-value computed between ME-MLS$_{AGA}$ and ME-MLS$_{FGAA}$.

The *hybrid* metrics measure both the convergence and the correct sampling of the target Pareto front. We consider in this work the Relative Hypervolume (RHV) hybrid metric. RHV is defined as the ratio of the volume covered by the Pareto front computed by the algorithm and the volume covered by the true Pareto front (in the objective functions space). Larger values of RHV indicate a closer convergence to the true Pareto front and a better sampling of non-dominated solutions in the calculated Pareto front.

Some of the aforementioned metrics require the true Pareto front of each problem instance to be computed, which is unknown for the studied ME-HCSP instances. For those cases, the true Pareto front was approximated by gathering all the non-dominated solutions computed using both ME-MLS variants considering the 30 independent executions performed for each algorithm.

In order to determine the significance of the comparison, a statistical analysis was performed over the results for each algorithm, metric, and problem instance solved. First, the Kolmogorov-Smirnov (K-S) test was applied to check whether the metric values follow a normal distribution or not. The values for the D statistic by the K-S test indicated that the results for ME-MLS$_{AGA}$ and ME-MLS$_{FGAA}$ are not normally distributed. As a consequence, the non-parametric Kruskal-Wallis statistical test was performed with a confidence level of 95%, to compare the distributions for ME-MLS$_{AGA}$ and ME-MLS$_{FGAA}$. For each metric and heterogeneity class, the best algorithm and the number of problem instances in which it is the best with 95% confidence (i.e. the computed pairwise p-value is below 5×10^{-2}) is also reported in the best$_{95\%}$ column. The result is emphasized in **bold** font when a given algorithm variant is always better than the other one in the 11 problem instances solved for each instance model, consistency type, heterogeneity class, and problem dimension.

Table 6.7 reports the computed results for the considered metrics for the 512×16 dimension instances. The reported results show that the ME-MLS$_{AGA}$ algorithm outperforms the ME-MLS$_{FGAA}$ algorithm in the ND metric results, but does not provide significant results on all of the eleven machine scenarios for any of the instance model, consistency type, and heterogeneity class. When comparing the IGD metric, the ME-MLS$_{AGA}$ algorithm outperforms ME-MLS$_{FGAA}$ algorithm on the computed results for the consistent and semi-consistent type of instances, while the ME-MLS$_{FGAA}$ algorithm prevails when considering the inconsistent type of instances. But again, without significant results on all the eleven machine scenarios. Similar results are reported for the Spread and RHV metrics, ME-MLS$_{AGA}$ slightly prevails when considering the consistent and semi-consistent type of instances, and ME-MLS$_{FGAA}$ when considering the inconsistent type of instances.

Table 6.8 reports the computed results for the considered metrics for the 1024×32 dimension instances. The results show that the ME-MLS$_{AGA}$ algorithm outperforms the ME-MLS$_{FGAA}$ algorithm in the ND metric for the computed results, this time with significant confidence for most of the consistent and semi-consistent type of instances. The ME-MLS$_{AGA}$ also significantly outperforms the ME-MLS$_{FGAA}$ when comparing the results computed by the IGD and the RHV metrics for the consistent and semi-consistent type of instances. On the contrary, the Spread metric shows a much even behavior, showing very similar results to the ones computed for the 512×16 dimension instances. It is worth noting that, so far, the ME-MLS$_{FGAA}$ algorithm was unable to significantly outperform the ME-MLS$_{AGA}$ algorithm in all of the eleven machine scenarios for any of the evaluated instance model, consistency type, and heterogeneity class,

Table 6.9 reports the computed results for the considered metrics for the 2048×64 dimension instances. The ME-MLS$_{AGA}$ algorithm continues to outperform the ME-MLS$_{FGAA}$ algorithm in the results computed for every considered metric for the consistent and semi-consistent type of instances. Further increasing the differences previously computed for the 1024×32 dimension instances.

The results demonstrate that ME-MLS$_{AGA}$ outperforms ME-MLS$_{FGAA}$ in terms of both efficacy and diversity metrics when solving consistent and semi-consistent type of instances, specially when solving the largest ME-HCSP dimension instances. On the contrary, even though ME-MLS$_{FGAA}$ computes slightly better results than ME-MLS$_{AGA}$ when solving the inconsistent type of instances, this difference is not significant and ME-MLS$_{FGAA}$ is unable to clearly outperform ME-MLS$_{AGA}$ in any case. Furthermore, because of the low number of computed non-dominated solutions (ND), neither of the ME-MLS algorithms are able to adequately sample the Pareto front of the inconsistent type of instances. Tables 6.10 and 6.11 report the average and standard deviation summary results for both ME-MLS variants, regarding the problem dimension for the efficacy and diversity metrics, respectively. The experiments demonstrate that, in average, the results computed by ME-MLS$_{FGAA}$ are competitive when solving the 512×16 dimension instances, but ME-MLS$_{FGAA}$ rapidly starts to lag behind in efficacy and diversity as the dimension of the instances increases. On average, ME-MLS$_{AGA}$ was able to find Pareto fronts with better diversity and covering properties than ME-MLS$_{FGAA}$ when tackling instances of dimensions 1024×32 and 2048×64, as it is demonstrated by the values of all the considered metrics.

Table 6.7: ME-MLS multi-objective metrics for the 512×16 dimension instances.

model	**consis-tency**	**hetero-geneity**	**ND** AGA	FGAA	$best_{95\%}$	**IGD** (normalized) AGA	FGAA	$best_{95\%}$
Ali	cons.	high high	8.33±2.47	4.11±1.43	AGA 10/11	1.00±0.27	1.44±0.18	AGA 6/11
		high low	8.02±2.65	4.20±1.39	AGA 8/11	1.00±0.15	1.46±0.33	AGA 5/11
		low high	6.67±2.42	3.82±1.20	AGA 10/11	1.00±0.18	1.52±0.28	AGA 8/11
		low low	2.84±1.54	2.49±1.12	FGAA 3/11	1.00±0.20	1.04±0.19	FGAA 1/11
	incons.	high high	2.06±1.15	2.20±1.21	FGAA 1/11	1.02±0.14	1.00±0.13	none
		high low	2.50±1.53	2.35±1.22	none	1.00±0.09	1.02±0.11	none
		low high	2.37±1.43	2.47±1.40	none	1.00±0.12	1.00±0.11	FGAA 1/11
		low low	1.60±0.87	1.53±0.77	none	1.00±0.09	1.04±0.12	FGAA 1/11
	semi.	high high	6.03±2.56	3.99±1.38	AGA 6/11	1.00±0.18	1.19±0.19	FGAA 4/11
		high low	5.91±2.72	4.07±1.39	AGA 4/11	1.00±0.12	1.23±0.22	AGA 3/11
		low high	4.64±2.11	3.48±1.29	AGA 6/11	1.00±0.13	1.05±0.16	AGA 2/11
		low low	2.50±1.21	2.21±1.00	AGA 2/11	1.00±0.17	1.05±0.15	FGAA 2/11
Braun	cons.	high high	6.63±3.06	3.75±1.50	AGA 7/11	1.00±0.17	1.42±0.25	AGA 5/11
		high low	6.18±3.30	3.36±1.50	AGA 8/11	1.00±0.27	1.45±0.29	AGA 6/11
		low high	6.64±3.03	3.84±1.56	AGA 7/11	1.00±0.16	1.28±0.26	AGA 5/11
		low low	6.02±3.10	3.21±1.21	AGA 7/11	1.00±0.25	1.48±0.21	AGA 7/11
	incons.	high high	2.14±1.36	2.19±1.26	none	1.00±0.18	1.04±0.16	FGAA 1/11
		high low	1.99±1.15	1.78±1.00	AGA 1/11	1.02±0.20	1.00±0.15	none
		low high	2.15±1.26	2.23±1.23	none	1.00±0.20	1.08±0.23	none
		low low	1.83±1.06	1.80±0.93	none	1.09±0.18	1.00±0.19	none
	semi.	high high	6.44±2.39	3.85±1.35	AGA 7/11	1.00±0.14	1.31±0.23	FGAA 4/11
		high low	6.62±2.35	3.66±1.27	AGA 9/11	1.00±0.25	1.52±0.15	AGA 7/11
		low high	6.75±2.38	3.95±1.33	AGA 10/11	1.00±0.13	1.28±0.23	AGA 5/11
		low low	6.02±2.39	3.75±1.22	AGA 7/11	1.00±0.21	1.35±0.20	AGA 5/11

model	**consis-tency**	**hetero-geneity**	**Spread** (normalized) AGA	FGAA	$best_{95\%}$	**RHV** AGA	FGAA	$best_{95\%}$
Ali	cons.	high high	1.00±0.04	1.07±0.09	AGA 5/11	0.85±0.06	0.79±0.07	AGA 6/11
		high low	1.00±0.05	1.04±0.06	FGAA 4/11	0.82±0.08	0.78±0.08	AGA 5/11
		low high	1.00±0.05	1.20±0.12	AGA 6/11	0.89±0.05	0.85±0.06	AGA 6/11
		low low	1.01±0.08	1.00±0.08	FGAA 2/11	0.82±0.09	0.81±0.08	none
	incons.	high high	1.00±0.01	1.00±0.01	FGAA 1/11	0.76±0.10	0.76±0.09	none
		high low	1.00±0.03	1.00±0.03	none	0.73±0.10	0.73±0.11	none
		low high	1.00±0.01	1.00±0.01	AGA 1/11	0.78±0.10	0.78±0.10	FGAA 1/11
		low low	1.00±0.03	1.00±0.04	FGAA 5/11	0.74±0.11	0.72±0.10	AGA 1/11
	semi.	high high	1.00±0.05	1.01±0.05	AGA 1/11	0.81±0.08	0.78±0.08	FGAA 4/11
		high low	1.00±0.10	1.04±0.09	AGA 2/11	0.79±0.10	0.76±0.09	AGA 4/11
		low high	1.00±0.08	1.05±0.07	FGAA 4/11	0.85±0.07	0.84±0.06	none
		low low	1.01±0.07	1.00±0.08	AGA 1/11	0.79±0.08	0.78±0.08	FGAA 2/11
Braun	cons.	high high	1.00±0.08	1.02±0.10	none	0.84±0.07	0.79±0.08	AGA 5/11
		high low	1.00±0.10	1.08±0.08	AGA 5/11	0.91±0.04	0.88±0.04	AGA 5/11
		low high	1.00±0.09	1.04±0.06	AGA 3/11	0.80±0.09	0.77±0.09	AGA 5/11
		low low	1.00±0.13	1.11±0.09	AGA 7/11	0.92±0.03	0.91±0.03	AGA 5/11
	incons.	high high	1.00±0.01	1.00±0.01	FGAA 1/11	0.78±0.10	0.77±0.09	FGAA 1/11
		high low	1.00±0.05	1.01±0.05	AGA 5/11	0.84±0.07	0.84±0.07	none
		low high	1.00±0.01	1.00±0.01	AGA 1/11	0.79±0.10	0.78±0.10	none
		low low	1.00±0.01	1.00±0.01	AGA 2/11	0.88±0.06	0.89±0.06	none
	semi.	high high	1.00±0.07	1.02±0.06	none	0.84±0.07	0.80±0.07	FGAA 4/11
		high low	1.00±0.05	1.10±0.08	AGA 5/11	0.90±0.04	0.87±0.04	AGA 5/11
		low high	1.00±0.05	1.05±0.06	AGA 5/11	0.83±0.06	0.81±0.06	AGA 4/11
		low low	1.00±0.05	1.14±0.11	AGA 5/11	0.91±0.04	0.90±0.04	AGA 2/11

Table 6.8: ME-MLS multi-objective metrics for the 1024×32 dimension instances.

model	consis-tency	hetero-geneity	ND			IGD (normalized)		
			AGA	FGAA	$best_{95\%}$	AGA	FGAA	$best_{95\%}$
Ali	cons.	high high	9.98±0.22	4.44±1.31	**AGA 11/11**	1.00±0.22	2.33±0.32	**AGA 11/11**
		high low	9.99±0.09	4.16±1.38	**AGA 11/11**	1.00±0.16	2.56±0.30	AGA 10/11
		low high	8.67±1.48	3.63±1.01	**AGA 11/11**	1.00±0.20	3.22±0.19	**AGA 11/11**
		low low	5.90±1.90	2.92±0.84	AGA 10/11	1.00±0.29	1.50±0.36	AGA 5/11
	incons.	high high	3.28±2.05	2.88±1.45	FGAA 1/11	1.00±0.13	1.05±0.10	AGA 1/11
		high low	3.17±2.03	2.68±1.36	none	1.00±0.12	1.05±0.16	none
		low high	2.50±1.48	2.39±1.29	AGA 1/11	1.00±0.09	1.08±0.14	none
		low low	1.62±0.81	1.60±0.83	none	1.00±0.16	1.03±0.14	none
	semi.	high high	9.85±0.60	4.26±1.32	**AGA 11/11**	1.00±0.20	2.01±0.19	AGA 10/11
		high low	9.96±0.26	4.52±1.20	**AGA 11/11**	1.00±0.16	2.00±0.22	AGA 10/11
		low high	5.73±1.83	3.42±0.98	**AGA 11/11**	1.00±0.12	1.44±0.20	AGA 6/11
		low low	3.59±1.59	2.67±1.00	AGA 5/11	1.00±0.21	1.08±0.17	AGA 2/11
Braun	cons.	high high	9.98±0.16	4.14±1.23	**AGA 11/11**	1.00±0.15	2.26±0.35	**AGA 11/11**
		high low	9.95±0.43	4.72±1.35	**AGA 11/11**	1.00±0.15	2.42±0.27	AGA 10/11
		low high	9.96±0.42	4.75±1.30	**AGA 11/11**	1.00±0.21	2.28±0.33	**AGA 11/11**
		low low	8.56±1.49	3.63±0.99	**AGA 11/11**	1.00±0.18	2.97±0.31	**AGA 11/11**
	incons.	high high	2.80±1.58	2.66±1.42	AGA 1/11	1.00±0.09	1.03±0.11	AGA 1/11
		high low	3.20±1.80	3.09±1.54	FGAA 1/11	1.00±0.10	1.05±0.11	none
		low high	3.13±1.83	2.89±1.37	AGA 1/11	1.00±0.12	1.11±0.11	AGA 1/11
		low low	2.81±1.71	2.44±1.37	AGA 1/11	1.02±0.15	1.00±0.14	none
	semi.	high high	9.90±0.58	4.53±1.13	**AGA 11/11**	1.00±0.22	2.13±0.35	AGA 9/11
		high low	9.85±0.71	4.91±1.20	**AGA 11/11**	1.00±0.12	2.20±0.18	AGA 9/11
		low high	9.91±0.50	4.83±1.15	**AGA 11/11**	1.00±0.15	2.07±0.30	**AGA 11/11**
		low low	6.61±1.74	3.52±0.96	**AGA 11/11**	1.00±0.16	1.94±0.29	AGA 8/11

model	consis-tency	hetero-geneity	Spread (normalized)			RHV		
			AGA	FGAA	$best_{95\%}$	AGA	FGAA	$best_{95\%}$
Ali	cons.	high high	1.00±0.07	1.29±0.13	AGA 9/11	0.86±0.05	0.75±0.07	AGA 10/11
		high low	1.00±0.08	1.17±0.13	AGA 7/11	0.84±0.05	0.72±0.08	AGA 10/11
		low high	1.00±0.05	1.39±0.14	AGA 10/11	0.92±0.03	0.81±0.05	**AGA 11/11**
		low low	1.00±0.07	1.00±0.07	AGA 2/11	0.90±0.04	0.89±0.04	AGA 3/11
	incons.	high high	1.00±0.03	1.00±0.03	none	0.75±0.11	0.74±0.10	AGA 1/11
		high low	1.00±0.04	1.00±0.04	AGA 1/11	0.76±0.10	0.75±0.10	none
		low high	1.00±0.02	1.00±0.03	AGA 1/11	0.77±0.09	0.75±0.09	none
		low low	1.00±0.02	1.00±0.02	FGAA 6/11	0.69±0.12	0.69±0.12	none
	semi.	high high	1.00±0.05	1.06±0.09	AGA 6/11	0.83±0.06	0.75±0.07	AGA 7/11
		high low	1.00±0.04	1.17±0.05	AGA 9/11	0.83±0.07	0.74±0.07	AGA 10/11
		low high	1.00±0.06	1.14±0.12	AGA 4/11	0.85±0.06	0.82±0.06	FGAA 4/11
		low low	1.00±0.06	1.00±0.08	none	0.84±0.09	0.82±0.10	FGAA 1/11
Braun	cons.	high high	1.00±0.05	1.21±0.13	AGA 7/11	0.86±0.05	0.77±0.06	AGA 10/11
		high low	1.00±0.06	1.32±0.15	AGA 8/11	0.85±0.05	0.74±0.07	AGA 8/11
		low high	1.00±0.05	1.36±0.11	AGA 10/11	0.86±0.05	0.76±0.06	AGA 10/11
		low low	1.00±0.04	1.40±0.14	AGA 10/11	0.91±0.03	0.82±0.05	**AGA 11/11**
	incons.	high high	1.00±0.02	1.00±0.02	FGAA 2/11	0.75±0.10	0.74±0.11	AGA 1/11
		high low	1.01±0.02	1.00±0.02	none	0.75±0.10	0.74±0.10	none
		low high	1.00±0.03	1.01±0.03	AGA 1/11	0.76±0.10	0.74±0.10	none
		low low	1.00±0.01	1.01±0.01	none	0.78±0.10	0.77±0.09	AGA 1/11
	semi.	high high	1.00±0.04	1.17±0.05	AGA 10/11	0.83±0.07	0.72±0.07	AGA 8/11
		high low	1.00±0.04	1.23±0.10	**AGA 11/11**	0.85±0.06	0.76±0.06	AGA 7/11
		low high	1.00±0.02	1.22±0.05	AGA 9/11	0.81±0.07	0.71±0.07	AGA 10/11
		low low	1.00±0.06	1.25±0.13	AGA 7/11	0.89±0.04	0.85±0.04	AGA 8/11

Table 6.9: ME-MLS multi-objective metrics for the 2048×64 dimension instances.

model	**consistency**	**heterogeneity**	**ND** AGA	**ND** FGAA	**ND** best$_{95\%}$	**IGD** (normalized) AGA	**IGD** (normalized) FGAA	**IGD** (normalized) best$_{95\%}$
Ali	cons.	high high	10.00±0.00	5.25±0.92	**AGA 11/11**	1.00±0.16	2.02±0.35	**AGA 11/11**
		high low	10.00±0.05	5.20±0.99	**AGA 11/11**	1.00±0.17	2.08±0.17	AGA 10/11
		low high	9.09±1.19	3.72±1.10	**AGA 11/11**	1.00±0.11	3.15±0.24	**AGA 11/11**
		low low	8.02±1.63	3.66±1.18	**AGA 11/11**	1.00±0.24	1.41±0.18	AGA 6/11
	incons.	high high	4.55±2.51	3.40±1.48	AGA 4/11	1.00±0.21	1.14±0.12	AGA 3/11
		high low	4.80±2.47	3.41±1.41	AGA 4/11	1.00±0.21	1.18±0.19	FGAA 3/11
		low high	3.08±1.58	2.88±1.38	AGA 1/11	1.00±0.23	1.02±0.22	none
		low low	1.34±0.64	1.35±0.63	none	1.06±0.19	1.00±0.18	none
	semi.	high high	10.00±0.05	4.78±1.16	**AGA 11/11**	1.00±0.15	2.74±0.33	**AGA 11/11**
		high low	10.00±0.05	4.61±1.10	**AGA 11/11**	1.00±0.15	2.96±0.44	**AGA 11/11**
		low high	7.81±1.75	3.40±0.99	**AGA 11/11**	1.00±0.22	2.38±0.31	AGA 9/11
		low low	3.22±1.71	2.79±1.19	AGA 4/11	1.12±0.18	1.00±0.16	none
Braun	cons.	high high	10.00±0.00	5.19±0.93	**AGA 11/11**	1.00±0.18	2.07±0.27	**AGA 11/11**
		high low	10.00±0.00	5.18±0.92	**AGA 11/11**	1.00±0.24	2.07±0.22	**AGA 11/11**
		low high	10.00±0.00	5.23±0.87	**AGA 11/11**	1.00±0.25	1.90±0.18	**AGA 11/11**
		low low	9.49±0.86	4.03±1.16	**AGA 11/11**	1.00±0.26	2.79±0.30	**AGA 11/11**
	incons.	high high	4.41±2.29	3.34±1.39	AGA 3/11	1.00±0.16	1.21±0.12	AGA 4/11
		high low	4.73±2.61	3.45±1.47	AGA 4/11	1.00±0.14	1.21±0.16	AGA 3/11
		low high	4.87±2.47	3.35±1.42	AGA 6/11	1.00±0.19	1.24±0.21	AGA 4/11
		low low	3.32±1.70	3.00±1.42	AGA 2/11	1.05±0.18	1.00±0.14	FGAA 1/11
	semi.	high high	10.00±0.00	4.58±1.15	**AGA 11/11**	1.00±0.25	2.43±0.13	**AGA 11/11**
		high low	10.00±0.05	4.68±1.09	**AGA 11/11**	1.00±0.17	2.82±0.27	AGA 10/11
		low high	10.00±0.05	4.89±0.99	**AGA 11/11**	1.00±0.23	2.83±0.22	**AGA 11/11**
		low low	7.70±1.75	3.42±0.99	**AGA 11/11**	1.00±0.28	2.23±0.32	**AGA 11/11**
model	**consistency**	**heterogeneity**	**Spread** (normalized) AGA	**Spread** (normalized) FGAA	**Spread** (normalized) best$_{95\%}$	**RHV** AGA	**RHV** FGAA	**RHV** best$_{95\%}$
Ali	cons.	high high	1.00±0.06	1.47±0.19	AGA 10/11	0.82±0.04	0.76±0.08	AGA 7/11
		high low	1.00±0.06	1.45±0.15	AGA 10/11	0.83±0.04	0.77±0.07	AGA 8/11
		low high	1.00±0.07	1.35±0.13	AGA 10/11	0.88±0.03	0.81±0.06	**AGA 11/11**
		low low	1.00±0.06	1.17±0.16	AGA 5/11	0.82±0.06	0.84±0.04	FGAA 7/11
	incons.	high high	1.01±0.05	1.00±0.06	none	0.79±0.09	0.77±0.09	AGA 3/11
		high low	1.01±0.04	1.00±0.05	FGAA 1/11	0.78±0.09	0.75±0.09	FGAA 3/11
		low high	1.00±0.04	1.01±0.03	FGAA 1/11	0.78±0.10	0.78±0.09	none
		low low	1.00±0.01	1.00±0.01	FGAA 7/11	0.67±0.12	0.69±0.11	none
	semi.	high high	1.00±0.04	1.51±0.11	**AGA 11/11**	0.82±0.06	0.65±0.09	AGA 10/11
		high low	1.00±0.05	1.53±0.12	**AGA 11/11**	0.82±0.06	0.64±0.09	AGA 9/11
		low high	1.00±0.05	1.21±0.11	AGA 9/11	0.87±0.05	0.80±0.06	AGA 8/11
		low low	1.02±0.07	1.00±0.06	none	0.73±0.11	0.77±0.12	AGA 4/11
Braun	cons.	high high	1.00±0.05	1.46±0.14	AGA 10/11	0.83±0.04	0.76±0.07	AGA 9/11
		high low	1.00±0.03	1.44±0.08	AGA 8/11	0.86±0.03	0.78±0.07	AGA 8/11
		low high	1.00±0.05	1.48±0.13	AGA 10/11	0.83±0.04	0.76±0.07	AGA 10/11
		low low	1.00±0.08	1.43±0.12	AGA 10/11	0.88±0.02	0.81±0.05	**AGA 11/11**
	incons.	high high	1.01±0.04	1.00±0.05	none	0.77±0.10	0.73±0.09	none
		high low	1.00±0.05	1.00±0.06	AGA 1/11	0.79±0.09	0.76±0.09	AGA 3/11
		low high	1.00±0.04	1.00±0.05	FGAA 1/11	0.78±0.10	0.73±0.10	AGA 4/11
		low low	1.00±0.02	1.01±0.03	FGAA 2/11	0.78±0.10	0.79±0.09	none
	semi.	high high	1.00±0.08	1.46±0.14	AGA 9/11	0.79±0.06	0.64±0.10	AGA 10/11
		high low	1.00±0.06	1.60±0.13	**AGA 11/11**	0.84±0.05	0.67±0.07	AGA 9/11
		low high	1.00±0.06	1.64±0.14	**AGA 11/11**	0.84±0.06	0.65±0.08	AGA 10/11
		low low	1.00±0.05	1.17±0.09	AGA 8/11	0.86±0.05	0.81±0.06	AGA 8/11

Table 6.10: Summary of ME-MLS multi-objective quality metrics.

dimension	**ND**		**IGD** (normalized)	
	AGA	FGAA	AGA	FGAA
512×16	4.70±2.23	3.10±0.87	1.00±0.08	1.20±0.13
1024×32	6.70±3.20	3.57±0.92	1.00±0.26	1.62±0.21
2048×64	7.35±2.90	3.95±0.99	1.00±0.27	1.71±0.27

Table 6.11: Summary of ME-MLS multi-objective diversity metrics.

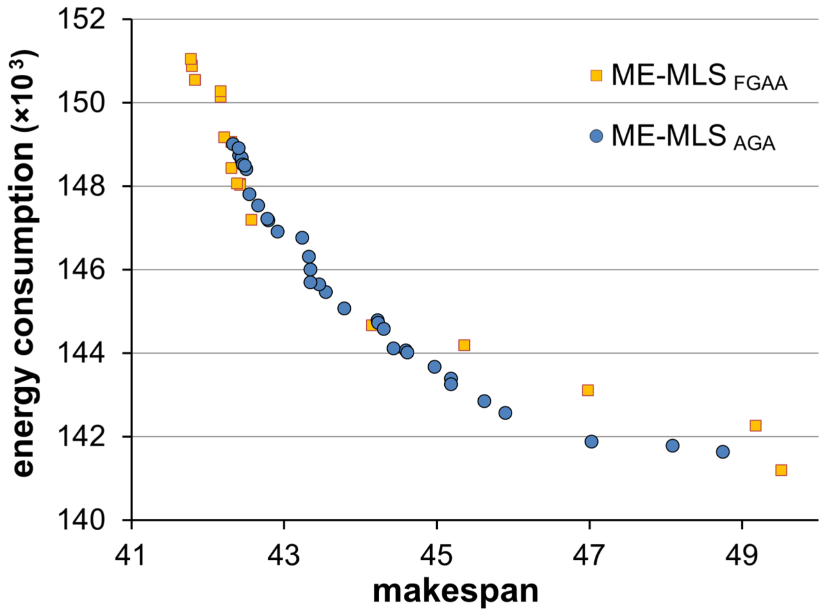
dimension	**Spread** (normalized)		**RHV**	
	AGA	FGAA	AGA	FGAA
512×16	1.00±0.09	1.04±0.06	0.83±0.05	0.81±0.05
1024×32	1.00±0.10	1.13±0.09	0.82±0.06	0.76±0.05
2048×64	1.00±0.09	1.25±0.16	0.81±0.05	0.75±0.06

Summarizing, the study indicates that the AGA replacement strategy is a useful choice to improve the efficacy and diversity of the schedules found by the ME-MLS algorithm, significantly improving over the FGAA replacement technique regarding the two main goals in multi-objective optimization.

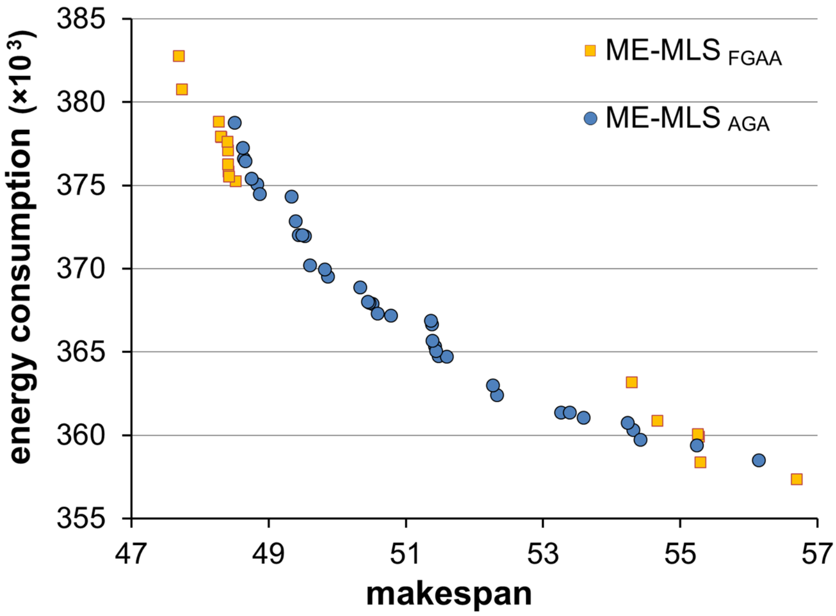
6.6.3 Summary

This section presented two different approaches for comparing the ME-MLS$_{AGA}$ and the ME-MLS$_{FGAA}$ algorithms: a solution quality comparison was presented in Section 6.6.1, and a multi-objective comparison was presented in Section 6.6.2.

When considering the consistency type of the tackled instances, the study shows there is a trade-off between objective function improvement and Pareto front sampling. The solution quality comparison analysis shows that both ME-MLS algorithms are able to archive the largest improvements when solving the inconsistent type of instances, achieving more moderated improvements for the consistent and semi-consistent type of instances. At the same time, the multi-objective comparison shows that neither ME-MLS algorithm is able to adequately sample the Pareto front of the inconsistent type of instances, while the ME-MLS$_{AGA}$ algorithm samples correctly the Pareto front of both consistent and semi-consistent type of instances. When comparing with each other, the solution quality comparison analysis demonstrated that the ME-MLS$_{FGAA}$ algorithm is able to compute extremal schedules with slightly better makespan and total energy consumption values than the ME-MLS$_{AGA}$ algorithm, specially for the largest ME-HCSP dimension instances. On the other hand, the multi-objective comparison demonstrated that the ME-MLS$_{AGA}$ algorithm is able to compute a more accurate and more diverse Pareto front than the ME-MLS$_{FGAA}$ algorithm, again specially for the largest ME-HCSP dimension instances. This results are exemplified in the Pareto front approximations samples in Figures 6.4 and 6.5. In these figures it can be clearly seen that the ME-MLS$_{FGAA}$ algorithm is able to compute better extremal schedules, while the ME-MLS$_{AGA}$ is able to compute schedules with better diversity and better covering properties.

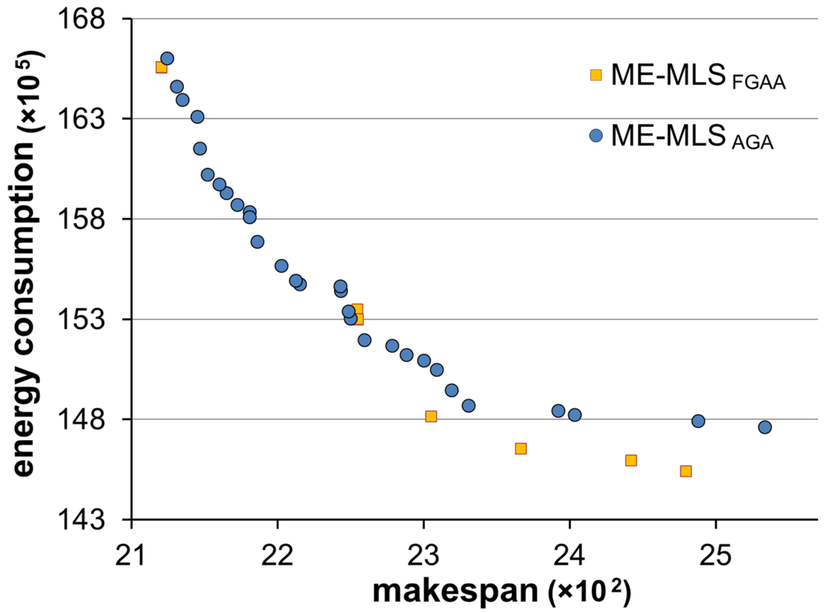


(a) Instance generated using the model by Ali et al. (2000) with a consistent structure, low task heterogeneity, and high machine heterogeneity

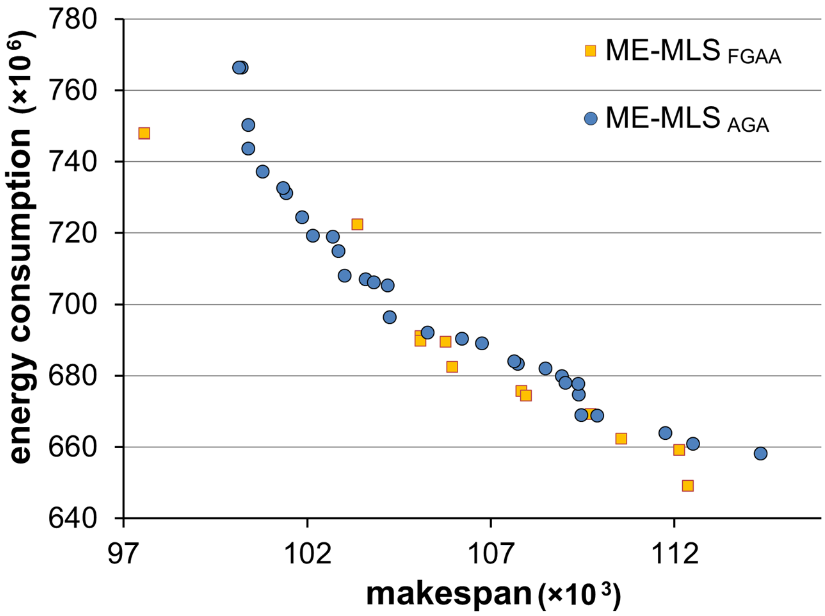


(b) Instance generated using the model by Braun et al. (2001) with a consistent structure, low task heterogeneity, and low machine heterogeneity

Figure 6.4: Approximated Pareto front computed by the ME-MLS algorithms after 30 independent executions solving two instances of dimension 1024×32.



(a) Instance generated using the model by Braun et al. (2001) with a semi-consistent structure, low task heterogeneity, and high machine heterogeneity



(b) Instance generated using the model by Braun et al. (2001) with a consistent structure, high task heterogeneity, and high machine heterogeneity

Figure 6.5: Approximated Pareto front computed by the ME-MLS algorithms after 30 independent executions solving two instances of dimension 2048×64.

6.7 Computational efficiency analysis

A speedup analysis was performed for both ME-MLS$_{AGA}$ and ME-MLS$_{FGAA}$ algorithms in order to study the behavior of the execution time of the algorithms when using different numbers of threads. The speedup evaluation was performed using a 1024×32 dimension instance, performing 30 independent executions of each version of the algorithm, and using a stopping criterion of 6 million iterations. Table 6.12 presents the speedup analysis results for ME-MLS$_{AGA}$ and ME-MLS$_{FGAA}$, respectively.

Table 6.12: Speedup values for the ME-MLS$_{FGAA}$ and ME-MLS$_{AGA}$ algorithms.

threads	**ME-MLS$_{FGAA}$**		**ME-MLS$_{AGA}$**	
	time (s)	speedup	time (s)	speedup
1	30.5±1.45	1.0	33.7±2.03	1.0
2	29.7±1.34	1.0	17.2±0.76	2.0
3	11.4±0.48	2.7	16.1±0.60	2.1
4	8.4±0.30	3.6	8.8±0.44	3.8
5	7.9±0.37	3.9	7.2±0.38	4.7
6	6.4±0.17	4.7	6.9±0.21	4.9
7	4.7±0.16	6.5	5.7±0.19	5.9
8	4.2±0.16	7.2	4.6±0.15	7.3
9	4.1±0.14	7.4	4.0±0.16	8.4
10	3.6±0.13	8.4	3.9±0.16	8.6
11	3.0±0.09	10.0	3.6±0.10	9.5
12	2.8±0.09	10.7	3.1±0.15	10.9
13	2.8±0.10	11.0	2.9±0.08	11.6
14	2.6±0.09	11.7	2.8±0.12	12.0
15	2.3±0.06	13.3	2.6±0.10	13.2
16	2.1±0.07	14.5	2.3±0.08	14.6
17	2.1±0.08	14.4	2.2±0.08	15.1
18	2.0±0.07	15.0	2.2±0.10	15.0
19	1.8±0.05	17.0	2.1±0.09	15.9
20	1.7±0.07	18.1	1.9±0.10	17.4
21	1.7±0.06	18.3	1.9±0.07	18.1
22	1.6±0.05	19.1	1.9±0.10	18.1
23	1.4±0.05	21.2	1.8±0.07	18.8
24	1.4±0.03	22.4	1.7±0.06	20.2

Figure 6.6 graphically summarizes the results of the speedup analysis when using up to 24 threads executing on the 24 cores available in the Magny-Cours experimental computing platform used.

The study shows that both ME-MLS$_{AGA}$ and ME-MLS$_{FGAA}$ algorithms have a near linear speedup behavior. The efficiency results in Table 6.12 demonstrates that ME-MLS$_{AGA}$ is the most efficient and scalable algorithm: it achieves a speedup of 22.4 when using 24 threads, while ME-MLS$_{FGAA}$ obtains a speedup of 20.2.

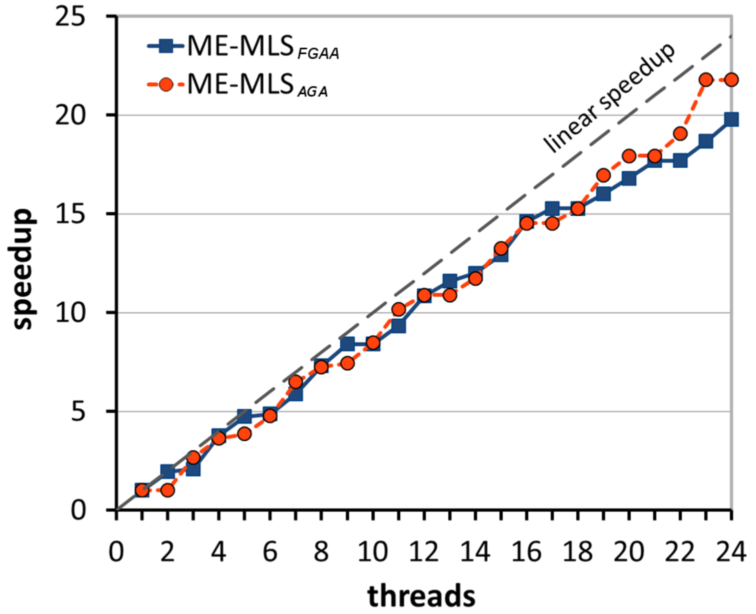


Figure 6.6: Speedup for the two variants of the ME-MLS algorithm.

6.8 Summary

This chapter presented a comprehensive experimental evaluation of the ME-MLS algorithm. The execution platform and the problem instances used for the experiments were presented. Two different techniques for computing baseline reference results were detailed. The first technique makes use of a linear programming relaxation technique, and provides a lower bound for each of the considered instances was calculated. The second technique makes use of four MinMin-based list-scheduling heuristics in order to compute accurate reference solutions. Two different versions of the ME-MLS algorithm were evaluated, the ME-MLS$_{AGA}$ and the ME-MLS$_{FGAA}$. When comparing to the best MinMin-based heuristic, both versions demonstrated to compute average makespan improvements of approximately 10%, and energy consumption improvements of approximately 6%. On the other hand, the ME-MLS$_{AGA}$ algorithm proved to compute a more accurate and evenly distributed Pareto front than the ME-MLS$_{FGAA}$ algorithm, specially for some type of instances. Finally, the computational efficiency analysis showed the ME-MLS$_{AGA}$ algorithm to have a better scalability behavior than the ME-MLS$_{FGAA}$, achieving an upper speedup value of 22.4 and 20.2 respectively.

Chapter 7

Scheduling very large problem scenarios

This chapter tackles the problem of solving very large problem scenarios for the single-objective HCSP. Previous work demonstrated that the rPALS algorithm is an accurate and efficient scheduler for tackling the HCSP (Nesmachnow et al., 2012b). Because of its proven capabilities, the rPALS was further extended with the use of the GPU architecture. The result of this further work is the gPALS algorithm: a hybrid rPALS-based algorithm designed for the CPU/GPU architecture.

First, a brief introduction to GPU computing is presented. After that, the gPALS algorithm is detailed, and the results of the experimental evaluation of the gPALS algorithm are reported and analyzed. Finally, the gPALS algorithm is compared with a hybrid CPU/GPU cellular EA proposed by Pinel et al. (2013), also for tackling the HCSP.

7.1 GPU computing

Graphics Processing Unit (GPU) devices were originally designed to exclusively perform graphic processing operations, allowing the *Central Process Unit* (CPU) to focus on the remaining general purpose computations. The GPU architecture specializes in massive parallel computation, and nowadays GPU devices are comprised of hundreds of parallel processing computing cores which are able to provide a considerably large aggregate computing power. In the last decade, the concept of *General-Purpose Graphics Processing Unit* (GPGPU) has gained increased popularity and hardware manufacturers have made a large effort in order to enable the execution of general purpose applications in the GPU architecture.

The first GPUs used for general-purpose computing were programmed using graphic-oriented programming interfaces such as OpenGL and DirectX (Fernando, 2004). Later, the applications for GPU were developed in assembly language for each device model, having very limited to none portability. More recently, high-level languages were developed to fully exploit the capabilities of the GPU architecture. In 2007, NVIDIA introduced the Compute Unified Device Architecture (CUDA) (Sanders and Kandrot, 2010), a software architecture for managing the GPU as a parallel computing device without requiring to map the data and the computation into a graphic-oriented programming interface.

CUDA extends the C language, and it is available since devices of the family GeForce 8 Series onwards. Three software layers are used in CUDA to communicate with the GPU (see Figure 7.1): a low-level hardware driver that performs the CPU-GPU data communications, a high-level runtime that provides the basic programming interfaces, and a set of general purpose libraries such as CUBLAS for linear algebra and CUFFT for Fourier transforms calculation.

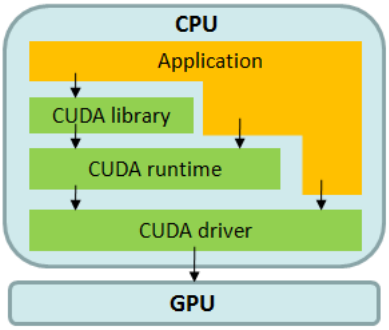


Figure 7.1: CUDA architecture.

The CUDA architecture is built around a scalable array of multiprocessors, each one having eight scalar processors, one multithreading unit, and a shared memory chip. The multiprocessors are able to efficiently create, manage, and execute a massive number of parallel threads. The threads are grouped into *blocks*, each block having up to 512 threads, and each block being executed in a single multiprocessor of the GPU device. Furthermore, the blocks are grouped into *grids*. Only one grid can be executed at a time, and each time a CUDA application calls a grid to be executed in the GPU device, all the blocks in the given grid are scheduled to be executed by the available multiprocessors.

When a multiprocessor receives a block to be executed, it groups the threads of the block into sets of 32 consecutive threads called *warps*. All the threads in a warp are able to execute a single instruction at a time, so the best efficiency is achieved when the 32 threads in the warp execute the same instruction over different data. Otherwise, the warp serializes the threads. When a block finishes its execution, the multiprocessor becomes idle and waits for the next block to be scheduled.

The threads are able to store and access application data using different memory spaces, the three most important of these memory spaces are: the per-thread *registers memory*, which is private to each thread; the per-block *shared memory*, which is shared between all the threads in the same block; and the per-application *global memory* of the GPU device, which is globally accessible by any thread in the application. The registers memory provides the fastest access rate of all the memory spaces, but it is also the one with the smallest storage capacity. The global memory has the slowest access rate, but the largest storage capacity. Finally, the shared memory provides a compromise between access rate and storage capacity. Minimizing the access to the slower memory spaces is a very important feature to achieve high efficiency in GPU computing. Figure 7.2 shows the previously described CUDA memory model.

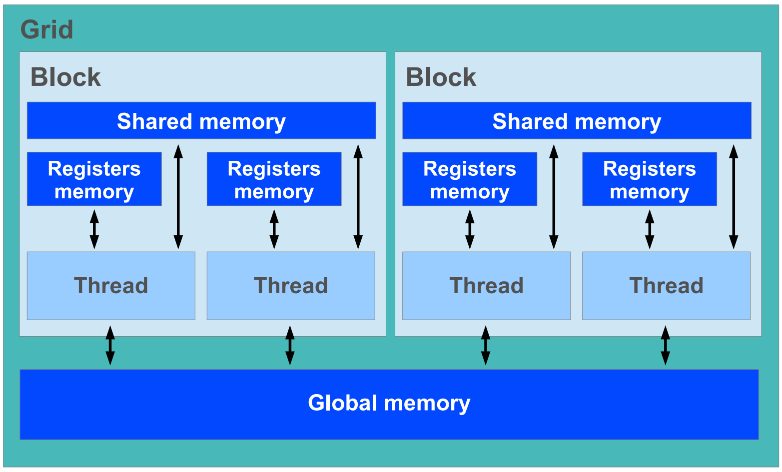


Figure 7.2: CUDA memory model.

7.2 gPALS: a rPALS-based GPU scheduler for the HCSP

The emergence of general purpose GPU computing has opened new promising lines of research. Indeed, this new technology will help to address larger problem scenarios in reasonable wall-clock times. This is precisely the main design goal of gPALS, a hybrid CPU/GPU parallel implementation of rPALS. In gPALS, the neighborhood evaluation of the embedded local search is massively parallelized and migrated to the CUDA architecture; while the higher level schema of the rPALS algorithm is executed in the traditional CPU architecture. Algorithm 16 shows the pseudo-code of the higher level schema of gPALS algorithm which is executed by the CPU architecture.

Algorithm 16 Pseudo-code of the gPALS algorithm for the HSCP

1: $s \leftarrow$ GenerateInitialSolution()
2: **while** stopping condition not met **do**
3: $\quad M \leftarrow$ Massively parallel neighborhood evaluation applied to s
4: $\quad s \leftarrow$ Apply the best movement from M
5: $\quad s \leftarrow$ Apply the rest of the movements in M in random order, only if they do not modify an already-modified machine
6: **end while**
7:
8: **return** s

The method starts by generating an initial schedule s (line 1). Each iteration, a neighborhood evaluation is performed in the GPU in order to find a set of candidate movements for improving current schedule s (line 3). The logic of this neighborhood evaluation is described in Algorithm 17. The neighborhood evaluation returns a set M with the best movements found by the neighborhood evaluation in the GPU. The movement $m^* \in M$, which improves the most the current schedule s, is always applied (line 4). The remaining movements in M are applied in random order as long as they do not undo a previously applied movement (line 5). Figure 7.3 presents the general schema of the gPALS algorithm.

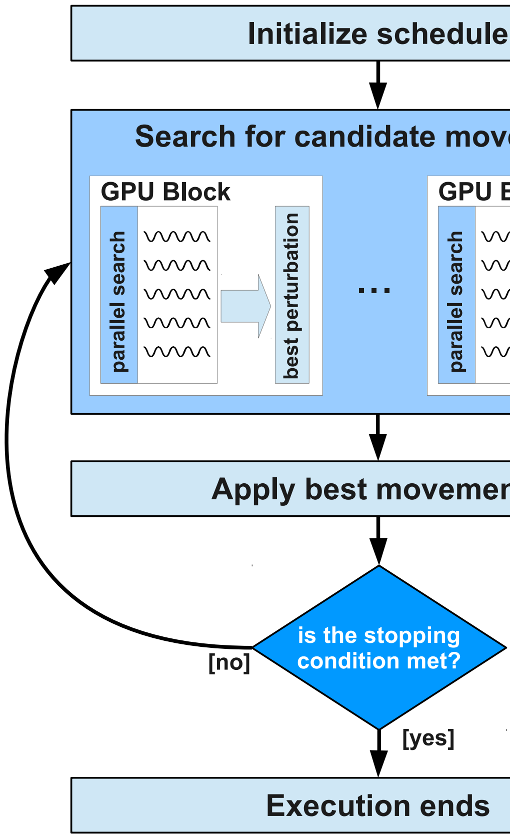


Figure 7.3: Schema of the gPALS algorithm.

The neighborhood evaluation on the GPU is organized in blocks, each block performs an independent neighborhood evaluation in a randomly selected neighborhood. During the search, all the threads in the same block collaborate with each other to find the best perturbation in the assigned neighborhood. Algorithm 17 presents the neighborhood search performed by each thread on the GPU in order to find a set of candidate perturbations for improving a given schedule.

First, each block selects a neighborhood structure: *swap* or *move* (line 2). These neighborhood structures are identical to the ones defined for the rPALS algorithm. After selecting the neighborhood structure, each thread in the block randomly selects the source and destination elements to modify (line 4 for the *swap* neighborhood structure, and lines 10–11 for the *move* neighborhood structure). Each thread evaluates one movement of the assigned neighborhood structure and computes the δ-value for it using the $CalculateDelta_{swap}(s, t_{src}, t_{dst})$ and $CalculateDelta_{move}(s, t_{src}, m_{dst})$ functions (line 5 and line 12). The movement and the δ-value of each thread are added to the set M_i, which is stored in the shared memory of the block i (line 7 and line 14). After that, a synchronization operation is performed among all the parallel threads within the block i (line 17). Once the threads of the block are synchronized, a parallel reduction operation is applied to find the best movement—i.e. the one with the best δ-value—(line 19). Finally, the best movement in M_i is returned to the higher level schema of gPALS in the CPU (see Algorithm 16).

Algorithm 17 Pseudo-code of the neighborhood search for the GPU

Require: s the current schedule to be improved
Shared $M_i \leftarrow \emptyset$ {shared among all the threads in the block i}
Shared $n \leftarrow$ Choose neighborhood structure (*swap* or *move*)
if n is *swap* **then**
  $t_{src}, t_{dst} \leftarrow$ Choose two random tasks ($t_{src} \neq t_{dst}$)
  $\delta \leftarrow \text{CalculateDelta}_{swap}(s, t_{src}, t_{dst})$
  if $\delta \geq 0$ **then**
    $M_i \leftarrow M_i \cup \{(t_{src}, t_{dst}, \delta)\}$
  end if
else if n is *move* **then**
  $t_{src} \leftarrow$ Choose a random task
  $m_{dst} \leftarrow$ Choose a random machine
  $\delta \leftarrow \text{CalculateDelta}_{move}(s, t_{src}, m_{dst})$
  if $\delta \geq 0$ **then**
    $M_i \leftarrow M_i \cup \{(t_{src}, m_{dst}, \delta)\}$
  end if
end if
SynchronizationBarrier() {synchronizes all threads in the block}

$m_i^{best} \leftarrow$ Parallel reduce M_i to find best movement in the block i
return m_i^{best}

Both $\mathit{CalculateDelta}_{swap}(s, t_{src}, t_{dst})$ and $\mathit{CalculateDelta}_{move}(s, t_{src}, m_{dst})$ functions are defined as their counterparts functions defined for the ME-rPALS local search, but considering only the makespan objective function (see Section 5.5.3).

Problem encoding

Two well-known structures for in-memory encoding of HCSP schedules were previously presented: the *task-oriented encoding* and the *machine-oriented encoding* (see Section 5.2).

When using the *machine-oriented encoding*, the HCSP schedule is stored in a $N \times M$-sized matrix. Lets suppose a 32-bit (4-byte) integer value (`int`) is used for task representation. Representing a schedule of dimension 32768×1024 would require a total memory of $(32768 \times 1024 \times 4)$ *bytes* $= 128$ Mb. On the other hand, the *task-oriented encoding* uses only a vector of size N for encoding a HCSP schedule, requiring only (32768×4) *bytes* $= 128$ Kb of memory for encoding a HCSP schedule.

The memory footprint of a schedule using the *machine-oriented encoding* is considerably larger than the one using the *task-oriented encoding.* This larger footprint can impact negatively in the algorithm performance, because of the time required to allocate and update such a large memory structure. Since the reduced execution time is a primary concern when designing the gPALS algorithm, the *task-oriented encoding* was chosen as the only in-memory encoding of HCSP schedules in the gPALS algorithm.

7.2.1 Initialization heuristics

The gPALS algorithm requires a method for generating an initial solution. Taking this into consideration, two different versions of gPALS have been devised depending on the method for generating the initial solution: *i)* gPALS$_{MCT}$, which uses MCT heuristic for generating the initial task schedule; and *ii)* gPALS$_{MMDD}$, which uses the pMinMin/DD heuristic for generating the initial task schedule.

The MCT heuristic was previously presented in Section 5.3, while the pMinMin/DD heuristic is presented next.

pMinMin/DD heuristic

The pMinMin/DD heuristic is a parallel MinMin heuristic with domain decomposition. The MinMin list-scheduling heuristic is considered to be one of the most accurate heuristics for solving the HCSP (Izakian et al., 2009a), but its design presents some scalability issues. The MinMin list-scheduling heuristic was previously introduced in Section 6.3.2 and it was shown that its execution time follows a $O(n^3)$ growth behavior.

In order to alleviate the computing requirement of the MinMin heuristic, the pMinMin/DD heuristic was devised by Canabé and Nesmachnow (2012). The pMinMin/DD applies a domain decomposition strategy, splitting the set of tasks T into p equally sized sub-groups $T_1, ..., T_p$. Then it computes the solution for each of the p sub-problems T_h applying the Min-Min heuristic and computing p sub-schedules $f_1, ..., f_p$. Finally, the p sub-schedules are aggregated into the final schedule $f = f_1 \cup ... \cup f_p$. Its naturally parallel design makes it simple to implement using a multithreading approach, computing each sub-domain schedule concurrently. The pseudo-code of the pMinMin/DD heuristic is presented in Algorithm 18.

Algorithm 18 Pseudo-code of each thread of the pMinMin/DD heuristic

Require: N number of tasks to be scheduled
  M number of available machines
  p number of sub-problems

$U_h \leftarrow T_h$ {set of unassigned tasks, $|T_h| = \frac{N}{p}$}
while $U_h \neq \emptyset$ **do**
  $L_h \leftarrow \emptyset$
  for each task $t_i \in U_h$ **do**
    for $j = 1 \rightarrow M$ **do**
      $ct_{ij} \leftarrow$ completion time of task t_i in machine m_j
      $L_h \leftarrow L_h \cup \{ct_{ij}\}$
    end for
  end for
  $ct^*_{ij} \leftarrow$ get the assignment with minimum completion time in L_h
  Assign task t_i to machine m_j
  Remove task t_i from L_h
end while

As the MinMin heuristic, the pMinMin/DD heuristic also presents three nested loops in its design, hence its execution complexity is in the order of $O\left(\frac{(N/p+1)\times N/p}{2} \times M\right) \sim O\left(\frac{N^2\times M}{2\times p^2}\right) \sim O\left(\frac{n^3}{2\times p^2}\right) \sim O(n^3)$.

In the long run, the pMinMin/DD heuristic presents the same complexity growth behavior as the MinMin heuristic, but a refined analysis shows a slower complexity growth thanks to the $1/p^2$ multiplier. Depending on the number of sub-problems p, this slower complexity growth could enable the pMinMin/DD heuristic to tackle much larger scenarios than the MinMin heuristic, while at the same time maintaining reduced execution times.

7.3 Experimental analysis of the gPALS algorithm

This section starts by introducing the set of HCSP instances and the computational platform used to evaluate the proposed gPALS algorithms. After that, the results computed by both gPALS algorithms are reported, and compared with the results computed by the MinMin heuristic. Finally, the results are summarized and compared with the ones computed using the cellular EA scheduler reported in the work by Pinel et al. (2013).

7.3.1 HCSP instances

To evaluate the proposed gPALS method, a specific set of 60 HCSP instances was used. These instances were randomly generated following the *range based* methodology proposed by Ali et al. (2000), and they were previously employed to evaluate the cellular EA scheduler for heterogeneous computing systems in the work by Pinel et al. (2013).

The proposed HCSP instances model realistic large-sized heterogeneous computing infrastructures. Three problem dimensions were studied in the experimental analysis of gPALS: (tasks×machines) 8192×256, 16384×512, and 32768×1024. These dimensions are larger than the ones usually tackled in the related literature, with the exception of the previously mentioned work by Pinel et al. (2013). For each comparison purposes, the gPALS was evaluated using the same instances proposed by Pinel et al. (2013). Pinel et al. (2013) generated a set of 20 instances for each dimension using the parametrization values suggested by Braun et al. (2001), all of which are publicly available.

7.3.2 Implementation details and execution platform

Both implementations of the gPALS algorithm were implemented in the C language, using the standard `stdlib` library and compiled with GNU C compiler 4.4.5 and the CUDA compiler 3.0. For efficiently generating pseudo-random numbers in the GPU architecture, the massively parallel *Mersenne Twister for Graphic Processors* (MTGP) library is used (Saito, 2010).

The experimental analysis was performed in a Bull B505 server with two six-core Intel Xeon CPU L5640 processor at 2.27GHz, 24 GB RAM, using a CentOS Linux operative system and a NVIDIA Tesla M2090 GPU device. The experimental platform is hosted as part of the HPC facility of the University of Luxembourg (platform website: `https://hpc.uni.lu/`).

7.3.3 Results and discussion

This section presents the experimental results obtained during the evaluation of the proposed gPALS methods. First, the *numerical efficacy* obtained when solving realistic HCSP instances is studied in detail, presenting a comparative analysis (regarding both the final results and the makespan evolution) between the two versions of gPALS and the MinMin heuristic. After that, the *parallel efficiency* of the proposed gPALS methods is studied, by comparing their execution times against the execution time required by the MinMin list scheduling heuristic. Finally, the summary presents a comparison of the makespan values and the computational efficiency with the results obtained using the cellular evolutionary scheduler proposed by Pinel et al. (2013).

Two different stopping criteria have been used in the gPALS algorithm during its experimental evaluation. In the numerical efficiency experiments, gPALS was configured for using a *fixed numerical effort* stopping criterion: the algorithm stops when 30 seconds of execution time have elapsed. This condition has been set to study the improvements in the solution quality achieved when running the proposed methods for a fixed execution time. On the other hand, in the parallel performance analysis of gPALS, a *fixed solution quality* stopping criterion was used: the algorithm stops when it reaches a task schedule with a lower makespan than the schedule computed by the MinMin heuristic. This experimental setting has been devised to analyze the acceleration in the required execution time when computing a solution with the same quality than the one computed by MinMin.

The gPALS method is comprised of several stages: *i) load*: loading the scheduling scenario from the instance file in disk; *ii) initialize*: computing the initial solution in CPU using either MCT or pMinMin/DD; *iii) initialize GPU*: initializing the GPU device; *iv) copy*: transferring the data from the CPU memory to the GPU memory; and *v) PALS*: performing the stochastic local search on CPU/GPU.

As an example, Figure 7.4 presents the absolute (values) and relative (bars) contributions of each stage to the total execution time for both gPALS versions. The time values were averaged per each problem dimension studied, when running the proposed algorithms until computing a schedule with similar makespan than the one computed by the MinMin list-scheduling heuristic. The *total time* indicates the wall-clock time for each gPALS method (i.e. the sum of the execution times for every stage).

Numerical efficiency

The numerical efficiency evaluation compares the total execution time and the accuracy of the two gPALS algorithms with the MinMin heuristic. For comparing the accuracy, the average and standard deviation of the computed makespan values are reported, considering the results from 30 independent executions performed for each algorithm and problem instance. For the execution time comparison, the time required to load the scheduling scenario from the instance file in disk is not considered, since it is exactly the same for all the compared algorithms. Thus, the reported execution time includes the time to compute the initial solution, the initialization of the GPU device, the time to transfer the data to the GPU, and the 30 wall-clock seconds to execute each PALS method.

In order to determine the statistical confidence of the results, the Kruskal-Wallis statistical test was performed to analyze the distributions of the results computed by each studied version of gPALS for each problem instance and dimension. The best results for each metric and problem instance are marked in **bold** font when the p-value computed in the correspondent pair-wise Kruskal-Wallis test is below 5×10^{-2} (meaning a statistical confidence of the results greater than 95%).

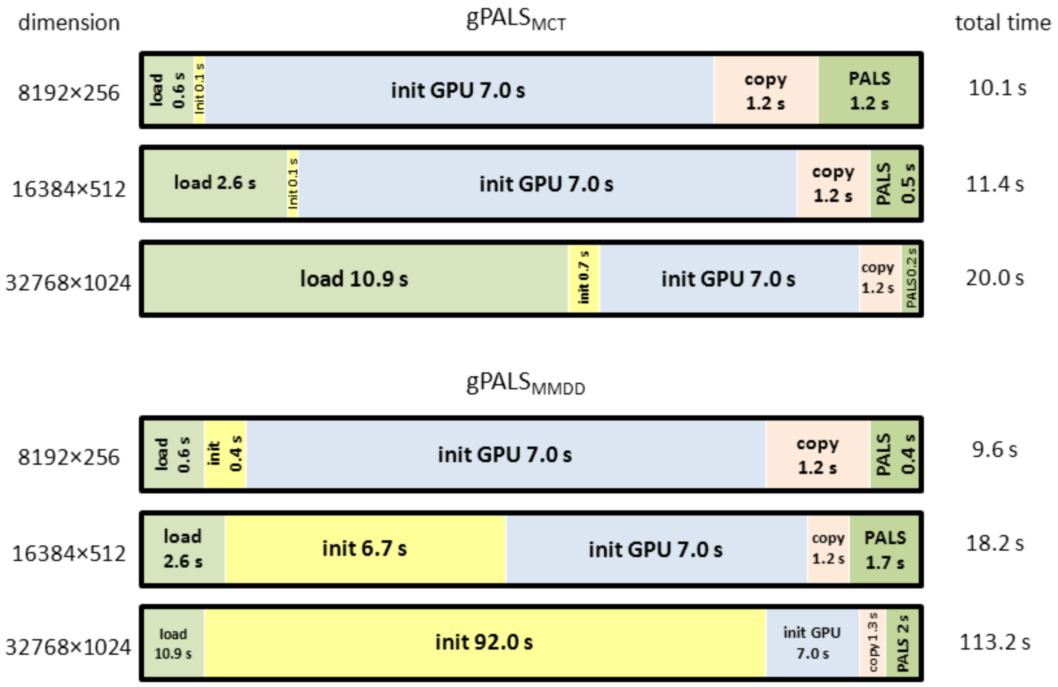


Figure 7.4: Contribution of each stage to the execution time for each gPALS version and each problem dimension studied.

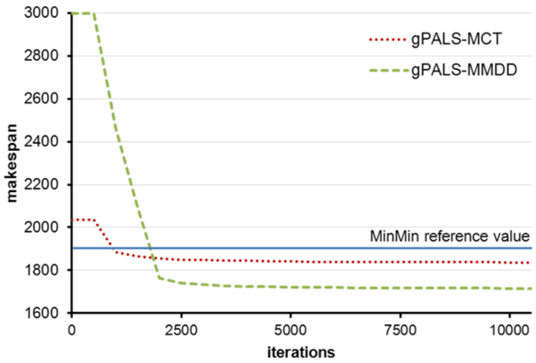
The experimental results in Tables 7.1, and 7.2 clearly point out that gPALS$_{MMDD}$ is the algorithm that computed the most accurate solutions. This situation occurs consistently for the three instances sizes tackled during the evaluation.

It is important to note the relevance of the makespan improvement values, given the experimental conditions. Maintaining a fixed execution time for PALS execution, the average makespan improvement values show that, the larger the instance dimension, the better the improvement. That is, for search spaces very much larger, the more the gPALS is able to improve over the MinMin result.

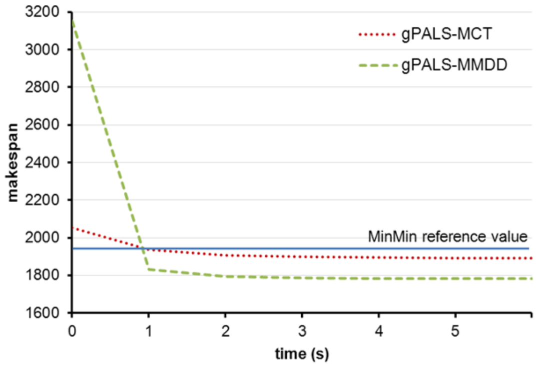
Figure 7.5 displays the evolution of the makespan values in a typical execution of both gPALS algorithms when solving a representative HCSP instance with dimension 8192×256. Two comparisons are presented, Figure 7.5a shows the makespan evolution in terms of the number of iterations of the algorithm, and Figure 7.5b shows the makespan evolution in terms of the wall-clock execution time of the GPU PALS algorithm. The makespan obtained by MinMin is also included in the graphics as a baseline for the comparison. The MCT heuristic computes a better initial solution than the pMinMin/D heuristic, but the gPALS$_{MCT}$ algorithm reaches a stagnation condition much faster than the gPALS$_{MMDD}$ heuristic. Despite the faster stagnation condition, the advantage provided by its initial solution allows the gPALS$_{MCT}$ algorithm to match the MinMin accuracy in around 1000 iterations, while the gPALS$_{MMDD}$ heuristic requires around 2000 iterations to match the same accuracy value. Both gPALS versions outperform MinMin after just about one single second of computation time. This shows the suitability of the proposed local search algorithms for efficiently addressing large-sized instances of the HCSP.

Table 7.1: Makespan and execution time comparison of MinMin and both gPALS versions for the 8192×256 dimension instances.

	instance	**makespan**			**execution time (s)**		
		MinMin	$gPALS_{MCT}$	$gPALS_{MMDD}$	MinMin	$gPALS_{MCT}$	$gPALS_{MMDD}$
8192×256 dimension instances	1	1845.2	1831.6±1.9	**1711.9±0.9**	15.0	38.6±0.7	39.4±0.5
	2	1889.9	1863.5±1.7	**1740.2±1.2**	15.0	38.7±0.4	39.3±0.4
	3	1894.3	1831.6±1.7	**1716.6±1.3**	14.8	38.7±0.4	39.4±0.5
	4	1890.1	1866.6±1.6	**1743.3±1.1**	14.9	38.8±0.6	39.4±0.5
	5	1859.6	1843.1±2.1	**1725.4±1.0**	14.9	38.7±0.6	39.5±0.6
	6	1863.4	1827.8±1.6	**1715.2±1.1**	14.9	39.0±0.6	39.3±0.7
	7	1897.3	1859.9±1.7	**1737.2±1.2**	14.9	38.8±0.6	39.5±0.5
	8	1874.5	1852.9±1.7	**1738.2±1.1**	15.3	38.9±0.6	39.4±0.5
	9	1871.5	1855.5±1.5	**1731.1±1.2**	15.0	38.7±0.5	39.3±0.5
	10	1865.9	1864.1±2.2	**1742.4±1.1**	15.0	38.8±0.6	39.3±0.5
	11	1840.7	1823.8±1.8	**1710.4±1.3**	14.8	38.6±0.6	39.3±0.7
	12	1867.3	1838.0±2.0	**1723.7±0.9**	14.9	38.7±0.7	39.2±0.6
	13	1895.4	1867.6±2.0	**1745.2±1.4**	14.9	38.4±0.6	39.3±0.5
	14	1884.8	1843.9±2.0	**1727.9±1.0**	14.9	38.8±0.5	39.1±0.6
	15	1851.0	1825.7±2.2	**1709.0±1.5**	14.9	38.7±0.7	39.2±0.6
	16	1846.3	1839.0±1.9	**1722.9±1.1**	15.1	38.8±0.5	39.3±0.6
	17	1874.7	1818.9±1.6	**1708.4±0.8**	15.0	38.6±0.6	39.2±0.7
	18	1862.8	1856.7±2.0	**1736.6±1.0**	14.6	38.6±0.5	39.2±0.6
	19	1892.5	1854.8±1.7	**1734.1±0.8**	14.9	38.5±0.7	39.4±0.5
	20	1869.0	1852.4±1.8	**1731.6±0.9**	14.9	38.8±0.6	39.2±0.6



(a) Makespan vs. iterations



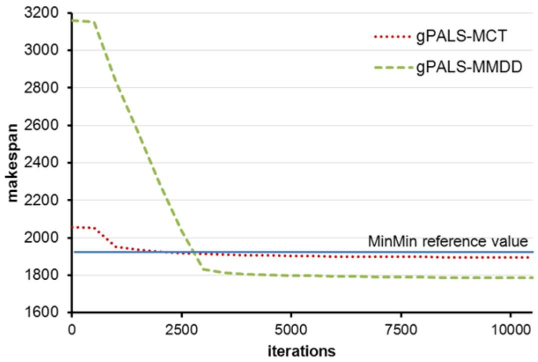
(b) Makespan vs. GPU wall-clock time

Figure 7.5: Evolution of the makespan value during a typical execution of $gPALS_{MCT}$ and $gPALS_{MMDD}$ for a 8192×256 dimension instance. The makespan computed by MinMin is shown as a reference baseline for the comparison.

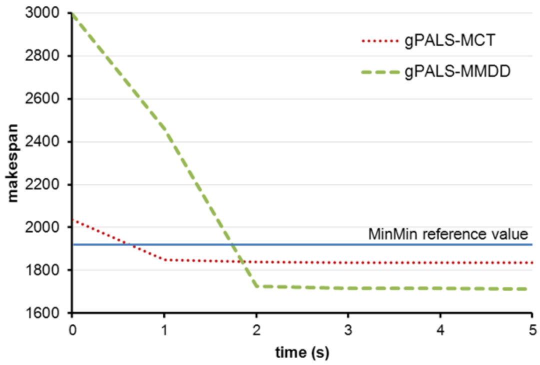
Table 7.2: Makespan and execution time comparison of MinMin and both gPALS versions for the 16384×512 and 32768×1024 dimension instances.

	instance	**makespan**			**execution time (s)**		
		MinMin	gPALS$_{MCT}$	gPALS$_{MMDD}$	MinMin	gPALS$_{MCT}$	gPALS$_{MMDD}$
16384×512 dimension instances	1	1934.1	1887.2±1.2	**1777.6±0.8**	110.7	40.7±0.6	47.5±0.6
	2	1940.5	1892.5±1.4	**1781.1±0.5**	110.6	40.8±0.7	47.2±0.6
	3	1949.0	1893.1±1.2	**1783.2±0.9**	110.6	40.6±0.5	47.4±0.5
	4	1922.0	1888.3±1.2	**1777.9±0.6**	110.7	40.8±0.6	47.5±0.5
	5	1904.1	1869.4±1.2	**1757.9±0.6**	111.2	40.6±0.7	47.4±0.5
	6	1901.7	1885.8±1.3	**1771.8±0.6**	111.4	40.8±0.6	47.4±0.6
	7	1945.5	1899.8±1.3	**1786.8±0.7**	111.3	40.9±0.6	47.6±0.5
	8	1903.8	1876.7±1.6	**1768.4±0.7**	111.3	40.6±0.8	47.5±0.6
	9	1937.3	1895.2±1.4	**1782.8±0.6**	111.1	40.8±0.6	47.3±0.5
	10	1935.7	1878.2±1.3	**1771.3±0.5**	110.7	40.8±0.5	47.3±0.8
	11	1937.4	1900.8±1.2	**1786.6±0.6**	110.8	40.7±0.8	47.4±0.6
	12	1916.2	1870.6±1.8	**1762.4±0.5**	110.9	40.8±0.6	47.4±0.6
	13	1911.8	1885.6±1.2	**1770.4±0.7**	110.7	40.7±0.6	47.5±0.6
	14	1927.6	1898.2±1.7	**1784.0±0.7**	110.8	40.6±0.7	47.7±0.7
	15	1944.4	1899.2±1.2	**1787.3±0.6**	110.5	40.5±0.7	47.4±0.5
	16	1939.5	1888.4±1.8	**1776.2±0.8**	110.8	40.5±0.7	47.4±0.6
	17	1933.6	1880.6±1.2	**1765.7±0.6**	111.3	40.8±0.5	47.5±0.5
	18	1929.5	1887.9±1.3	**1776.5±0.7**	111.3	40.8±0.7	47.4±0.6
	19	1910.8	1879.2±1.1	**1765.5±0.6**	111.2	40.7±0.8	47.5±0.5
	20	1941.0	1893.0±1.4	**1780.8±0.6**	111.1	40.7±0.7	47.5±0.6

	instance	**makespan**			**execution time (s)**		
		MinMin	gPALS$_{MCT}$	gPALS$_{MMDD}$	MinMin	gPALS$_{MCT}$	gPALS$_{MMDD}$
32768×1024 dimension instances	1	1996.2	1940.6±1.8	**1847.5±2.0**	839.8	49.6±0.7	139.7±7.2
	2	1979.0	1933.2±1.5	**1840.9±2.4**	838.8	49.5±0.7	137.3±6.0
	3	1980.7	1933.5±1.9	**1837.3±1.9**	842.5	49.8±0.9	137.8±8.0
	4	1982.8	1942.6±1.7	**1846.8±2.2**	841.7	49.8±0.7	139.3±6.8
	5	1971.9	1930.4±1.9	**1834.1±2.6**	845.3	49.6±0.6	139.2±5.8
	6	1973.4	1927.2±2.1	**1832.4±2.4**	834.6	49.3±0.6	137.5±8.2
	7	1991.0	1939.7±1.9	**1844.6±2.1**	837.4	49.6±0.6	137.3±6.6
	8	1991.8	1936.5±1.9	**1838.2±2.6**	843.2	49.4±0.7	138.9±6.9
	9	1994.8	1942.9±1.7	**1848.1±2.6**	838.4	49.5±0.7	128.3±6.8
	10	1997.1	1936.2±1.8	**1838.6±2.4**	839.3	49.6±0.6	136.7±6.7
	11	1975.8	1929.0±1.6	**1832.0±1.6**	840.4	49.6±0.7	128.3±8.0
	12	1974.2	1925.0±1.8	**1828.2±2.1**	838.7	49.5±0.7	138.6±7.0
	13	1978.6	1928.4±1.5	**1834.7±2.0**	843.1	49.6±0.8	128.3±8.7
	14	1988.1	1939.8±2.0	**1840.7±2.1**	841.8	49.5±0.8	138.0±8.0
	15	1972.1	1931.5±2.0	**1834.5±1.9**	844.0	49.7±0.7	128.3±5.5
	16	1979.3	1926.9±1.8	**1830.5±3.0**	834.9	49.5±0.6	139.5±7.0
	17	1991.8	1930.4±1.6	**1831.0±2.7**	837.6	49.5±0.6	135.8±8.1
	18	1986.8	1937.5±1.5	**1841.6±2.2**	842.7	49.5±0.7	138.4±8.4
	19	1975.2	1927.5±2.0	**1829.8±2.0**	838.8	49.7±0.6	137.7±6.3
	20	1991.7	1932.3±2.0	**1836.3±1.9**	840.0	49.4±0.6	140.6±5.0



(a) Makespan vs. iterations



(b) Makespan vs. GPU wall-clock time

Figure 7.6: Evolution of the makespan value during a typical execution of gPALS$_{MCT}$ and gPALS$_{MMDD}$ for a 16384×512 dimension instance. The makespan computed by MinMin is shown as a reference baseline for the comparison.

Figure 7.6 shows the evolution of the makespan values with respect to the number of iterations, and the execution time of the gPALS algorithm for a representative 16384×512-sized instance. All the previous claims hold as well, but requiring the gPALS algorithm more iterations and longer execution time, to match the MinMin heuristic makespan value.

Figure 7.7 graphically summarizes the average makespan improvements over the MinMin results for the two versions of gPALS and each problem dimension studied in this article.

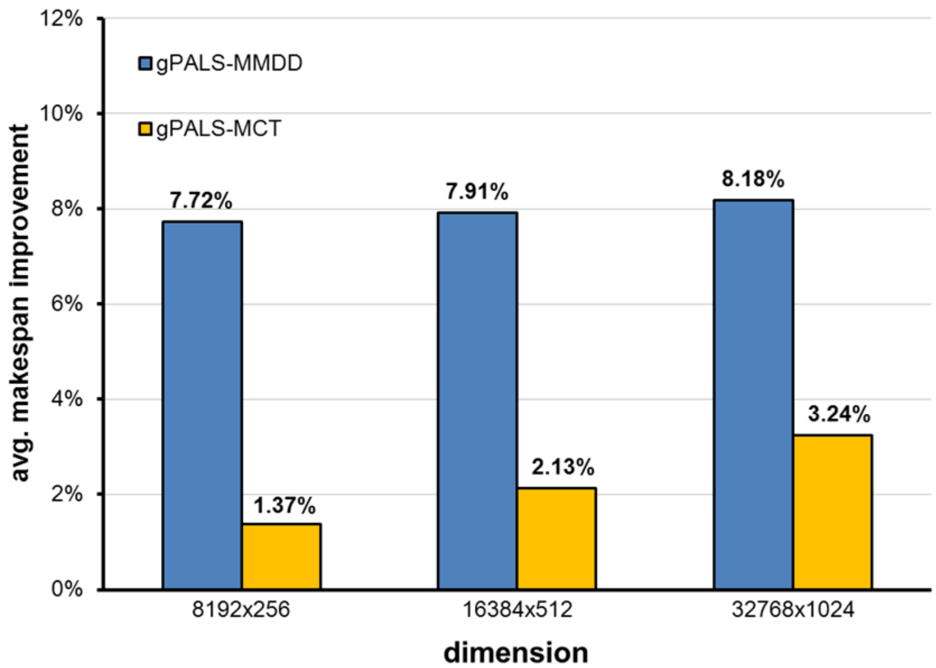


Figure 7.7: Average makespan improvements over MinMin for the two versions of gPALS.

Table 7.3: Wall-clock time (in seconds) required for each gPALS algorithm to compute a schedule with the same makespan than the schedule computed by MinMin, and compared with the MinMin average execution time.

inst.	**8192×256 dimension** (MinMin avg.: 15 s)		**16384×512 dimension** (MinMin avg.: 111 s)		**32768×1024 dimension** (MinMin avg.: 840 s)	
	gPALS$_{MCT}$	gPALS$_{MMDD}$	gPALS$_{MCT}$	gPALS$_{MMDD}$	gPALS$_{MCT}$	gPALS$_{MMDD}$
1	**5.50±0.17**	5.65±0.02	**7.30±0.06**	14.26±0.06	**16.24±0.09**	105.69±0.09
2	**5.15±0.02**	5.64±0.03	**7.26±0.03**	14.29±0.17	**16.22±0.09**	105.67±0.09
3	**4.89±0.02**	5.65±0.02	**7.15±0.26**	14.28±0.06	**16.26±0.10**	105.71±0.10
4	**5.20±0.04**	5.65±0.06	**7.54±0.04**	14.26±0.06	**16.26±0.11**	105.71±0.11
5	**5.38±0.07**	5.64±0.02	**7.52±0.04**	14.24±0.06	**16.20±0.09**	105.65±0.09
6	**5.05±0.03**	5.69±0.07	**8.39±0.15**	14.24±0.05	**16.21±0.10**	105.66±0.10
7	**5.03±0.02**	5.65±0.03	**7.35±0.03**	14.26±0.05	**16.26±0.12**	105.71±0.12
8	**5.22±0.03**	5.67±0.03	**7.76±0.05**	14.25±0.06	**16.21±0.11**	105.66±0.11
9	**5.37±0.06**	5.67±0.04	**7.38±0.04**	14.23±0.05	**16.25±0.12**	105.70±0.12
10	7.60±1.10	**5.67±0.05**	**7.19±0.03**	14.26±0.08	**16.22±0.10**	105.67±0.10
11	**5.33±0.05**	5.68±0.03	**7.49±0.03**	14.23±0.06	**16.17±0.10**	105.62±0.10
12	**5.11±0.02**	5.68±0.07	**7.34±0.02**	14.22±0.06	**16.24±0.12**	105.69±0.12
13	**5.16±0.02**	5.66±0.03	**7.77±0.05**	14.27±0.05	**16.22±0.13**	105.67±0.13
14	**5.00±0.02**	5.67±0.02	**7.64±0.04**	14.29±0.13	**16.25±0.14**	105.70±0.14
15	**5.16±0.03**	5.69±0.02	**7.34±0.04**	14.25±0.06	**16.21±0.10**	105.66±0.10
16	5.97±0.47	**5.65±0.02**	**7.26±0.03**	14.30±0.08	**16.25±0.13**	105.70±0.13
17	**4.91±0.02**	5.67±0.03	**7.24±0.03**	14.23±0.06	**16.21±0.12**	105.66±0.12
18	6.49±0.90	**5.66±0.02**	**7.38±0.03**	14.25±0.04	**16.22±0.11**	105.67±0.11
19	**5.03±0.02**	5.64±0.04	**7.61±0.05**	14.29±0.07	**16.24±0.10**	105.69±0.10
20	**5.35±0.05**	5.67±0.05	**7.30±0.06**	14.29±0.07	**16.25±0.13**	105.70±0.13

Parallel performance

Table 7.3 compares the wall-clock time of gPALS$_{MCT}$ and gPALS$_{MMDD}$ against the MinMin execution time, for the proposed HCSP instances. In the acceleration experiments, the reported wall-clock time accounts for the whole execution time, including the time required to load the instance from persistent storage into main memory.

For the following experiments, both gPALS methods use the *fixed solution quality* stopping criterion. This stopping criterion stops the gPALS algorithm execution when it computes a solution makespan value which at least matches the makespan value computed by the MinMin heuristic. This stopping criterion is considered in order to analyze the acceleration in the execution time to compute a solution with the same quality than the one computed by MinMin.

The first clear claim regarding the wall-clock times presented in Table 7.3, is that both gPALS versions match the quality of the MinMin computed solution in significantly reduced execution time. The larger the instance, the higher the reduction in the execution times. This can be explained by the ill $O(n^3)$ execution time growth presented by the MinMin heuristic. Indeed, MinMin requires roughly 15, 110, and 840 seconds to build a solution for 8192×256, 16384×512, and 32768×1024 instances, respectively, whereas gPALS$_{MCT}$ and gPALS$_{MMDD}$ need 8, 12, and 20 seconds, and 9, 18, and 110 seconds, respectively. These differences can be clearly seen in Figure 7.8, which displays the average execution time improvements over all the instances of the same size reached by gPALS$_{MCT}$ and gPALS$_{MMDD}$.

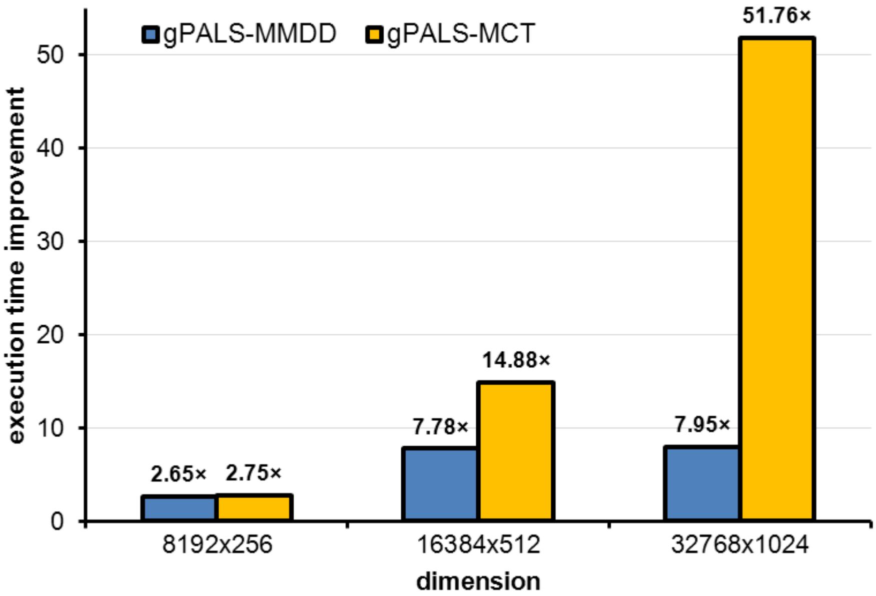


Figure 7.8: Average execution time improvements of gPALS$_{MCT}$ and gPALS$_{MMDD}$ with respect to MinMin.

The execution time improvements reported in Figure 7.8 refers to the *acceleration* or the *reduction in the execution time of an algorithm that runs in a parallel computing platform* (in this case the gPALS algorithm executing in CPU/GPU) *with respect another one that executes sequentially* (in this case, the MinMin heuristic executing in CPU).

For the smallest instance considered, the two gPALS algorithms perform roughly the same, with execution time improvements of 2.78× and 2.65×. However, as the dimension of the tackled instance increases, MinMin requires more time to complete, i.e. it does not scale well. On the contrary, the gPALS algorithms do scale properly, specially gPALS$_{MCT}$, which has been able to reach an execution time improvement of 51.76× for the largest dimension instance.

This significant difference in the execution time improvement factor for the largest instances of both gPALS methods is due to the computational time required by the initialization heuristic. As previously stated, MCT is a fast method which present a $O(n^2)$ execution time growth, while *pMinMin/DD* presents a $O\left(\frac{n^3}{p^2}\right)$ execution time growth, taking longer to build a solution and reducing the execution time improvement of gPALS$_{MMDD}$.

Summary: comparison against MinMin

Table 7.4 summarizes the average improvements over the makespan values computed using the MinMin list-scheduling heuristic, and the execution time acceleration over MinMin for each gPALS version and each instance dimension tackled.

The results in Table 7.4 demonstrate that the studied methods offer different trade-off solutions for the scheduling problem in heterogeneous computing environments. The gPALS$_{MMDD}$ algorithm is clearly the best scheduling algorithm for the 8192×256 dimension, with 7.72% of makespan improvement and 2.65× acceleration over MinMin. For the largest dimensions instances the choice is not so clear.

Table 7.4: Average makespan improvements and acceleration over MinMin for both gPALS versions.

dimension	avg. makespan improvement (over MinMin)		acceleration (over MinMin)	
	gPALS$_{MCT}$	gPALS$_{MMDD}$	gPALS$_{MCT}$	gPALS$_{MMDD}$
8192×256	1.37%±0.83%	**7.72%±0.73%**	2.78×	2.65×
16384×512	2.13%±0.53%	**7.91%±0.49%**	14.88×	7.78×
32768×1024	3.24%±0.30%	**8.18%±0.32%**	51.76×	7.95×

Table 7.5: Average makespan improvements over MinMin comparison of both gPALS versions against the cellular EA by Pinel et al. (2013).

dimension	avg. makespan improvement (over MinMin)		
	gPALS$_{MCT}$	gPALS$_{MMDD}$	cellular EA
8192×256	1.37%±0.83%	**7.72%±0.73%**	6.75%±0.73%
16384×512	2.13%±0.53%	**7.91%±0.49%**	6.05%±0.57%
32768×1024	3.24%±0.30%	**8.18%±0.32%**	5.19%±0.34%

On one hand, the gPALS$_{MCT}$ algorithm computes slightly better schedules significantly faster when compared with the MinMin scheduler—up to **51.76**× for dimension 32768×1024—. On the other hand, the gPALS$_{MMDD}$ algorithm computes significantly better schedules slightly faster than the MinMin scheduler, computing schedules with up to **8.18%** makespan reductions over MinMin with an acceleration of 7.95×.

Comparison against the cellular EA by Pinel et al. (2013)

Tables 7.5 and 7.6 present a comparison of both gPALS versions against the results reported by Pinel et al. (2013) when tackling the same problem using a cellular EA. Table 7.5 compares the average makespan values and Table 7.6 compares the average execution time for each method.

The results in Table 7.5 shows that gPALS$_{MMDD}$ clearly improves upon the state-of-the-art algorithms from the literature. These improvements have emerged not only in the quality of the computed schedules, but specially in the execution time required to compute them.

Table 7.6: Average execution time comparison of both gPALS versions against the cellular EA by Pinel et al. (2013).

dimension	**avg. execution time** (s)		
	gPALS$_{MCT}$	gPALS$_{MMDD}$	cellular EA
8192×256	**38.7±0.6**	39.3±0.6	1630.3±5.6
16384×512	**40.7±0.6**	47.4±0.6	4382.3±16.4
32768×1024	**49.6±0.7**	136.3±7.1	8088.3±58.3

Indeed, gPALS$_{MMDD}$ has achieved an increasing improvement rate over MinMin that grows up to 8.18% for the largest instance considered in this work (32768×1024). As discussed above, the largest the instance, the higher the improvement. The cellular EA, however, has reached its best improvement over MinMin in the smaller instance (8192×256) and the trend is just the opposite: the largest the instance, the smaller the improvement. It is worth noting that the cellular EA has clearly outperformed gPALS$_{MCT}$ (though the differences get very tight as the instances become larger).

Regarding the execution times reported in Table 7.6, the benefits of any of the two gPALS versions are conclusive. Both methods have an execution times in the range of a few dozen seconds (136 seconds at most), whereas the cellular EA requires 1630 seconds (more than 27 minutes) and 134 minutes (more than 2 hours) for the smallest instance and the largest instances, respectively. In all cases, the execution time of gPALS is always below 2.5% the execution time of the cellular EA.

7.4 Summary

This chapter introduced the topic of GPU computing, and presented gPALS, an hybrid CPU/GPU implementation of a randomized local search procedure for addressing large-size instances of the HCSP. A thorough experimental analysis was performed in order to evaluate the numerical efficiency and the parallel performance of the algorithm. The experimental analysis showed that the gPALS algorithm is able to outperform the Min-Min list-scheduling heuristic, both in accuracy and in execution time. The numerical efficiency analysis reported quality improvements of up to 8.17%, when using the pMin-Min/D initialization heuristic. The parallel performance analysis demonstrated that the proposed algorithm, using the MCT initialization heuristic, is able to provide acceleration rates of up to 51.6× when comparing with the MinMin heuristic. Finally, the accuracy and execution time of both gPALS versions were compared with the cellular EA proposed by Pinel et al. (2013). The results of this comparison showed the gPALS$_{MMDD}$ algorithm as the best compromise algorithm, computing the most accurate solutions and reporting very competitive execution times.

Chapter 8

Conclusions and future work

This chapter presents the conclusions of the work on addressing scheduling problems on heterogeneous computing environments in a reduced execution time. It also briefly details the main lines of future work for improving the current results.

8.1 Conclusions

This thesis tackled the problem of scheduling tasks in a heterogeneous computing environment in reduced execution times, considering both the schedule length and the total energy consumption as the optimization objectives.

In the last decade, heterogeneous computing systems have emerged as useful providers of the computing power needed to solve complex problems arising in many areas of application. Since their emergence, heterogeneous computing systems have become larger and larger mainly because of the ever demanding scientific community, and thanks to the fast increase of computing power and the rapid development of high-speed networking. Recently, energy consumption has become a major concern in large data centers. Processors are the main consumers of energy in such systems, and frequently they also offer the most flexible energy management mechanisms, by applying dynamic voltage scaling (DVS), dynamic power management, slack sharing and reclamation, etc. (Khan and Ahmad, 2009; Kim et al., 2007; Zhu et al., 2003). Reducing processors consumption is a great challenge, and researchers currently focus on the development of energy-aware scheduling algorithms for HC systems (Lee and Zomaya, 2009).

In order to model this reality, a formulation for the *Makespan-Energy Heterogeneous Computing Scheduling Problem* (ME-HCSP) was presented, based on the well-known *Heterogeneous Computing Scheduling Problem* (HCSP). The ME-HCSP describes the problem which arises in heterogeneous computing environments (such as computing grids or clusters) where both the energy consumption of the infrastructure, and the task schedule length, are critical. The ME-HCSP was categorized in the scheduling theory using the notation by Graham et al. (1979), and two different computing models in the related literature were detailed. Different techniques for tackling scheduling problems, such as enumerative algorithms, linear programming based algorithms, list-scheduling algorithms, and metaheuristic algorithms, were surveyed in the related literature.

Metaheuristic algorithms proved to be very accurate methods for solving scheduling problems, usually outperforming other methods reviewed in the literature. The efficacy and efficiency of local search metaheuristics make them the most appealing methods for rapidly and accurately solving optimization problems.

An exhaustive survey of recent works tackling energy-aware scheduling problems was presented. The survey shows that only a few recent works used a true multi-objective approach, most of the works avoid this kind of approach and instead introduce simplifications for tackling the problem as a single-objective problem.

The lack of true multi-objective approaches motivated the design of a true multi-objective ME-HCSP scheduling algorithm, considering both accuracy and efficiency. The efficiency is a capital issue when designing tasks scheduling algorithms for computing environments. The scheduling procedure must be able to rapidly assign incoming tasks to idle computing resources, otherwise the computing system would end up wasting computing cycles. The designed algorithm for tackling such problem, the ME-MLS, is a population-based local search metaheuristic. It achieves the much needed efficiency with a multithreading design which exploits the aggregated computing power of modern multi-core architectures. The ME-MLS, is based on Pareto dominance, and it works on a population of non-dominated schedules that are iteratively improved by applying the ME-rPALS local search algorithm. The ME-rPALS local search algorithm is a critical part of the ME-MLS, and was specifically designed for the ME-MLS. The ME-rPALS is based on the PALS (Alba and Luque, 2007) and rPALS (Nesmachnow et al., 2012b), two local search algorithms with proven results when tackling the HCSP. Two different variants of the ME-MLS were implemented, the ME-MLS$_{FGAA}$ algorithm and the ME-MLS$_{AGA}$, the former using a very simple archiving method designed for the ME-MLS, and the latter using the AGA archiving method (Knowles and Corne, 2003).

For the experimental evaluation of the ME-MLS algorithm, a lower bound for the problem instances was computed using a linear programming relaxation technique, and four different MinMin-based list-scheduling heuristics were devised for computing baseline reference values. In order to demonstrate the efficient performance of the algorithm, a reduced time stopping criterion of only 10 seconds was set in the experimental evaluation, allowing the ME-MLS to perform almost online scheduling. Both algorithm were evaluated using a large set of ME-HCSP instances representing small- and medium-sized heterogeneous computing systems. The efficacy and efficiency of both algorithms was evaluated and the results were statistically compared with each other. Comparing with each other, the experimental evaluation shows that ME-MLS$_{AGA}$ offers a better overall performance than ME-MLS$_{FGAA}$. Comparing with the proposed MinMin-based algorithms, the experimental analysis shows that both ME-MLS algorithms outperforms the best MinMin-based heuristic for every instance, with average improvements of up to 11.2% for the makespan and up to 6.8% for the energy consumption. A performance analysis was performed on both ME-MLS algorithms, and the results shows that the two ME-MLS variants have a promising scalability behavior.

The ME-MLS algorithm was evaluated using instances comprised of up to 2048 tasks and 64 machines. In order to further escalate the dimension of the tackled problem instances, the GPU architecture was explored. For initially addressing this challenge the gPALS, a hybrid CPU/GPU massively parallel local search method, was designed for solving the single-objective HCSP.

The gPALS is based on the ME-rPALS local search, previously proposed for the ME-MLS algorithm, but modified in order to take advantage of the additional computing power provided by the GPU architecture. The gPALS was evaluated solving instances comprised of up to 32768 tasks and 1024 machines in less than 30 seconds of execution time. Two different versions of the gPALS algorithm were proposed, depending on the initialization algorithm. The gPALS$_{MCT}$ makes use of the MCT heuristic for its initialization, and the gPALS$_{MMDD}$ makes use of the pMinMin/D heuristic. Both versions were statistically compared with each other, with the well-known MinMin heuristic, and with the cellular EA proposed by Pinel et al. (2013).

The experimental evaluation of the gPALS algorithm demonstrates that both gPALS implementations are able of compute better makespan values than MinMin in all of the studied instances. Comparing with each other, the gPALS$_{MMDD}$ algorithm proved to be the best method between the two gPALS implementations, obtaining significant improvements (up to 8.18%) with respect to MinMin. These reductions in the makespan obtained by gPALS$_{MMDD}$ have taken a wall clock time of 30 seconds, which represent a factor of almost 8× in the computational efficiency with respect to the MinMin scheduler. The gPALS$_{MCT}$ computed schedules significantly faster than both gPALS$_{MMDD}$ and MinMin, achieving execution time improvements up to 51.76× with respect to MinMin, but obtaining improvements of only up to 3.24%. Comparing with the cellular EA proposed by Pinel et al. (2013), the gPALS$_{MMDD}$ also is able to compute solutions with improved quality in much reduced execution time. This demonstrates that the new gPALS$_{MMDD}$ algorithm is an accurate and very efficient scheduler for the proposed HCSP instances.

8.2 Future work

The aggregate computing power provided by the GPU architecture seems promising, but the additional time required to initialize the GPU device make it unusable for small- and medium-sized scenarios. Hence, the pure CPU-based algorithmic solution should not be discarded, and two lines of future work presents for improving the proposed schedulers: the CPU-based scheduler, and the hybrid CPU/GPU scheduler.

For the CPU-based line of work, the main effort is focused on improving the ME-MLS algorithm, to make it more efficient, more accurate, and to increase the diversity of the non-dominated solutions computed by the algorithm. Regarding the ME-MLS accuracy improvement, work is needed for integrating the proposed ME-MLS algorithm into well-known multi-objective EA. The multi-objective EA would provide the diversity mechanisms required to compute higher quality Pareto solutions.

For the GPU-based line of work, there is much more work to do, as the GPU is a much younger architecture. The main effort is focused on designing ME-gPALS, a multi-objective version of gPALS for the energy-aware ME-HCSP. Then, as in the CPU-based line of work, the ME-gPALS should be integrated into a hybrid CPU/GPU multi-objective EA.

To improve the experimental evaluation of the CPU-based and the GPU-based schedulers, a line of work involves comparing the computed results with the results computed by well-known multi-objective EA, such as NSGA-II (Deb et al., 2002) and SPEA2 (Zitzler et al., 2001). Although these methods are much slower than the ME-MLS, this comparison should provide a reference framework for the computed Pareto front.

The final line of research relates to the ME-HCSP formulation. Work is to be done in order to formulate a problem which models a more realistic scenario. The EMC computing model proposed by Nesmachnow et al. (2012a) should be fully adopted in order to consider multi-core architecture machines in the computing environment. An online version of the problem should be formulated, and online instances which consider tasks arrival times should be constructed. And finally, parallel complex tasks should be considered, in order to model tasks such as multithreading applications, and MPI applications.

Appendix A

Publications by the author

This appendix presents a list of all the publications by the author.

International Journals

- S. Nesmachnow and S. Iturriaga. Multiobjective grid scheduling using a domain decomposition based parallel micro evolutionary algorithm. *International Journal of Grid and Utility Computing (IJGUC)*, 2013. Accepted on January 2012, to appear.
- S. Iturriaga, S. Nesmachnow, B. Dorronsoro, and P. Bouvry. Energy efficient scheduling in heterogeneous systems with a parallel multiobjective local search. *Computing and Informatics Journal (CAI)*, 2013a. Accepted on November 2012, to appear.
- S. Iturriaga, S. Nesmachnow, F. Luna, and E. Alba. A parallel local search in CPU/GPU for scheduling independent tasks on large heterogeneous computing systems. *Journal of Parallel and Distributed Computing (JPDC)*, 2013b. Submitted on January 2013, pending acceptance.

International conferences

- S. García, S. Iturriaga, and S. Nesmachnow. Scientific computing in the Latin America-Europe GISELA grid infrastructure. In *Proceedings of the 4th High-Performance Computing Latin America Symposium (HPCLatAm)*, JAIIO '11, pages 48–62, Córdoba City, Argentina, 2011
- S. García, S. Iturriaga, S. Nesmachnow, M. da Silva, M. Galnarés, G. Rodriguez, and G. Usera. Developing parallel applications in the GISELA grid infrastructure. In *Proceedings of the Joint GISELA-CHAIN Conference*, COMETA '12, pages 9–16, Mexico City, Mexico, 2012.
- S. Nesmachnow and S. Iturriaga. Multiobjective Scheduling on Distributed Heterogeneous Computing and Grid Environments Using a Parallel Micro-CHC Evolutionary Algorithm. In *Proceedings of the 6th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pages 134–141, Barcelona, Spain, 2011

- S. Iturriaga, S. Nesmachnow, and B. Dorronsoro. A Multithreading Local Search For Multiobjective Energy-Aware Scheduling In Heterogeneous Computing Systems. In *Proceedings of the 26th European Conference on Modelling and Simulation (ECMS)*, pages 497–503, Koblenz, Germany, 2012a. ISBN 978-0-9564944-4-3.

- S. Iturriaga, S. Nesmachnow, F. Luna, and E. Alba. A parallel online GPU scheduler for large heterogeneous computing systems. In *Proceedings of the 5th High-Performance Computing Latin America Symposium (HPCLatAm)*, JAIIO '12, Buenos Aires, Argentina, 2012b.

- S. Iturriaga and S. Nesmachnow. Solving Very Large Optimization Problems (Up to One Billion Variables) with a Parallel Evolutionary Algorithm in CPU and GPU. In *Proceedings of the Sixth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pages 267–272, Victoria, Canada, 2012. ISBN 978-1-4673-2991-0.

- S. Iturriaga, P. Ruiz, S. Nesmachnow, B. Dorronsoro, and P. Bouvry. A Parallel Multi-objective Local Search for AEDB Protocol Tuning. In *Proceedings of the 16th International Workshop on Nature Inspired Distributed Computing*, in the 27th IEEE/ACM International Parallel & Distributed Processing Symposium, Boston, Massachusetts, USA, 2013c. Accepted on February 2013, to appear.

International workshops

- S. Iturriaga, S. Nesmachnow, and C. Tutté. Metaheuristics for multiobjective energy-aware scheduling in heterogeneous computing systems. In *EU/Metaheuristics Meeting Workshop (EU/ME)*, Copenhaguen, Denmark, 2012c.

National conferences

- S. Iturriaga, D. Garat, and G. Moncecchi. Restauración automática de acentos ortográficos en adverbios interrogativos. In *Proceedings of the XII Argentine Symposium on Artificial Intelligence (ASAI)*, JAIIO '11, pages 108–119, Córdoba City, Argentina, 2011.

- S. Iturriaga and S. Nesmachnow. Bi-objective scheduling in heterogeneous grid computing systems using a parallel micro evolutionary algorithm. In *Proceedings of the XLIII Brazilian Symposium of Operational Research (SBPO)*, pages 1618–1629, São Paulo, Brazil, 2011.

Bibliography

E. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization.* John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1997. ISBN 0471948225.

A. Abraham, H. Liu, C. Grosan, and F. Xhafa. Nature inspired meta-heuristics for grid scheduling: single and multi-objective optimization approaches. *Metaheuristics for Scheduling in Distributed Computing Environments*, pages 247–272, 2008.

A. Agarwal, M. Ahmed, A. Berman, B. Caron, A. Charbonneau, D. Deatrich, R. Desmarais, A. Dimopoulos, I. Gable, L. Groer, R. Haria, R. Impey, L. Klektau, C. Lindsay, G. Mateescu, Q. Matthews, A. Norton, W. Podaima, D. Quesnel, R. Simmonds, R. Sobie, B. S. Arnaud, C. Usher, D. Vanderster, M. Vetterli, R. Walker, and M. Yuen. GridX1: A Canadian computational grid. *Future Generation Computer Systems*, 23 (5):680–687, 2007. ISSN 0167-739X.

M. Aggarwal, R. D. Kent, and A. Ngom. Genetic Algorithm Based Scheduler for Computational Grids. In *Proceedings of the 19th International Symposium on High Performance Computing Systems and Applications*, HPCS '05, pages 209–215, Washington, DC, USA, 2005. IEEE Press. ISBN 0-7695-2343-9.

E. Alba and B. Dorronsoro. *Cellular Genetic Algorithms*, volume 42 of *Operations Research/Computer Science Interfaces.* Springer-Verlag, 2008. ISBN 978-0-387-77609-5.

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 the 7th European Conference on Evolutionary Computation in Combinatorial Optimization*, volume 4446 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, 2007.

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*, HCW '00, pages 185–, Washington, DC, USA, 2000. IEEE Press. ISBN 0-7695-0556-2.

K. Anshumali, T. Chappell, W. Gomes, J. Miller, N. Kurd, and R. Kumar. Circuit And Process Innovations to Enable High-Performance, and Power and Area Efficiency on the Nehalem and Westmere Family of Intel processors. *Intel Technology Journal*, 14 (3), 2010.

J. Apodaca, D. Young, L. Briceno, J. Smith, S. Pasricha, A. A. Maciejewski, H. J. Siegel, S. Bahirat, B. Khemka, A. Ramirez, and Y. Zou. Stochastically robust static resource allocation for energy minimization with a makespan constraint in a heterogeneous computing environment. In *Proceedings of the 9th IEEE/ACS International Conference* *on Computer Systems and Applications*, AICCSA '11, pages 22–31, Washington, DC, USA, 2011. IEEE Press. ISBN 978-1-4577-0475-8.

L. A. Barroso. The Price of Performance. *Queue*, 3(7):48–53, Sept. 2005. ISSN 1542-7730.

S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi. Characterization of scientific workflows. In *Proceedings of the 3rd Workshop on Workflows in Support of Large-Scale Science*, WORKS '08, pages 1–10, 2008.

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

C. Blum and A. Roli. Hybrid Metaheuristics: An Introduction. In C. Blum, M. Aguilera, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics*, volume 114 of *Studies in Computational Intelligence*, pages 1–30. Springer-Verlag, 2008. ISBN 978-3-540-78294-0.

M. Bohlouli and M. Analoui. Grid-HPA: Predicting Resource Requirements of a Job in the Grid Computing Environment. *International Journal of Computer Science*, 3(3): 137–141, 2008. ISSN 13064428.

D. Bozdag, U. Catalyurek, and F. Ozguner. A task duplication based bottom-up scheduling algorithm for heterogeneous environments. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium*, IPDPS '06, pages 12–, 2006.

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.

J. Brownlee. *Clever Algorithms: Nature-Inspired Programming Recipes*. Lulu Enterprises, 2011. ISBN 9781446785065.

T. Burd, T. Pering, A. Stratakos, and R. Brodersen. A dynamic voltage scaled microprocessor system. In *Proceedings of the Solid-State Circuits Conference (ISSCC)*, Digest of Technical Papers, pages 294–295, 466. IEEE Press, 2000.

R. Buyya. *Economic-based Distributed Resource Management and Scheduling for Grid Computing*. PhD thesis, Monash University, 2002.

M. Canabé and S. Nesmachnow. Parallel implementations of the MinMin heterogeneous computing scheduler in GPU. *CLEI Electronic Journal*, 15(3):1–11, 2012.

K. Christodoulopoulos, V. Gkamas, and E. A. Varvarigos. Statistical Analysis and Modeling of Jobs in a Grid Environment. *Journal of Grid Computing*, 6(1):77–101, 2008.

C. Coello, G. Lamont, and D. Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer-Verlag, Secaucus, NJ, USA, 2006. ISBN 0387332545.

C. Coello, C. Dhaenens, and L. Jourdan. *Advances in Multi-Objective Nature Inspired Computing*. Springer-Verlag, 1st edition, 2010. ISBN 364211217X, 9783642112171.

P. Conway and B. Hughes. The AMD Opteron northbridge architecture. *Micro, IEEE*, 27(2):10–21, 2007.

T. T. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms.* MIT Press, Cambridge, MA, USA, 1990. ISBN 0-262-03141-8.

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

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2): 182–197, 2002. ISSN 1089-778X.

M. L. Dertouzos. Control Robotics: The Procedural Control of Physical Processes. In *Proceedings of the International Federation for Information Processing (IFIP) Congress*, pages 807–813, 1974.

C. Diaz, M. Guzek, J. Pecero, G. Danoy, P. Bouvry, and S. Khan. Energy-aware fast scheduling heuristics in heterogeneous computing systems. In *Proceedings of the International Conference on High Performance Computing and Simulation (HPCS)*, pages 478–484, 2011.

M. Dorigo and L. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1 (1):53–66, 1997. ISSN 1089-778X.

S. Droste, T. Jansen, and I. Wegener. Upper and Lower Bounds for Randomized Search Heuristics in Black-Box Optimization. *Theory of Computing Systems*, 39(4):525–544, 2006. ISSN 1432-4350.

R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the 6th International Symposium on Micro Machine and Human Science*, MHS '95, pages 39–43, 1995.

X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, pages 13–23, New York, NY, USA, 2007. ACM Press. ISBN 978-1-59593-706-3.

T. A. Feo and M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, Apr. 1989. ISSN 0167-6377.

R. Fernando, editor. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics.* Addison-Wesley, Inc., Boston, 2004.

S. Forrest and M. Mitchell. Relative Building-Block Fitness and the Building Block Hypothesis. In L. D. Whitley, editor, *Proceedings of the 2nd Workshop on Foundations of Genetic Algorithms*, pages 109–126. Morgan Kaufmann Publishers, Inc., 1992. ISBN 1-55860-263-1.

I. Foster and C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers, Inc., San Francisco, CA, USA, 2003. ISBN 1558609334.

R. Fourer, D. M. Gay, and B. W. Kernighan. A modeling language for mathematical programming. *Management Science*, 36(5):519–554, 1990. ISSN 0025-1909.

R. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. Lima, et al. Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet. In *Proceedings of the 7th Heterogeneous Computing Workshop*, HCW '98, pages 184–199. IEEE, 1998.

R. Friese, T. Brinks, C. Oliver, H. Siegel, and A. Maciejewski. Analyzing the Trade-offs Between Minimizing Makespan and Minimizing Energy Consumption in a Heterogeneous Resource Allocation Problem. In *Proceedings of the 2nd International Conference on Advanced Communications and Computation*, INFOCOMP '12, pages 81–89, Venice, Italy, 2012.

S. García, S. Iturriaga, and S. Nesmachnow. Scientific computing in the Latin America-Europe GISELA grid infrastructure. In *Proceedings of the 4th High-Performance Computing Latin America Symposium (HPCLatAm)*, JAIIO '11, pages 48–62, Córdoba City, Argentina, 2011.

S. García, S. Iturriaga, S. Nesmachnow, M. da Silva, M. Galnarés, G. Rodriguez, and G. Usera. Developing parallel applications in the GISELA grid infrastructure. In *Proceedings of the Joint GISELA-CHAIN Conference*, COMETA '12, pages 9–16, Mexico City, Mexico, 2012.

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

S. K. Garg, C. S. Yeo, A. Anandasivam, and R. Buyya. Environment-conscious scheduling of HPC applications on distributed Cloud-oriented data centers. *Journal of Parallel and Distributed Computing*, 71(6):732–749, 2011. ISSN 0743-7315. Special Issue on Cloud Computing.

M. Gendreau and J.-Y. Potvin. *Handbook of Metaheuristics*. Springer-Verlag, 2nd edition, 2010. ISBN 1441916636, 9781441916631.

C. Glasner and J. Volkert. A Three-Phase Adaptive Prediction System of the Run-Time of Jobs Based on User Behaviour. In *Proceedings of the International Conference on Complex, Intelligent and Software Intensive Systems*, CISIS '09, pages 886–891, 2009.

F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986. ISSN 0305-0548.

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, USA, 1989. ISBN 1-55860-066-3.

R. Graham, E. L. J. Lawler, and A. Kan. Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. *Annals of Discrete Mathematics*, 5: 287–326, 1979.

J. Hamilton. Cooperative expendable micro-slice servers (CEMS): low cost, low power servers for internet-scale services. In *Proceedings of the 4th Biennial Conference on Innovative Data Systems Research (CIDR)*, California, USA, 2009.

S. Hartmann. A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics (NRL)*, 49(5):433–448, 2002. ISSN 1520-6750.

T. Hey and A. E. Trefethen. The UK e-Science Core Programme and the Grid. *Future Generation Computer Systems*, 18(8):1017–1031, 2002. ISSN 0167-739X.

H. Hoos and T. Stützle. *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, USA, 2004. ISBN 1558608729.

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

A. Iosup and D. Epema. Grid Computing Workloads. *Internet Computing*, 15(2):19–26, 2011. ISSN 1089-7801.

A. Iosup, C. Dumitrescu, D. Epema, H. Li, and L. Wolters. How are Real Grids Used? The Analysis of Four Grid Traces and Its Implications. In *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, GRID '06, pages 262–269, Washington, DC, USA, 2006. IEEE Press. ISBN 1-4244-0343-X.

S. Iturriaga and S. Nesmachnow. Bi-objective scheduling in heterogeneous grid computing systems using a parallel micro evolutionary algorithm. In *Proceedings of the XLIII Brazilian Symposium of Operational Research (SBPO)*, pages 1618–1629, São Paulo, Brazil, 2011.

S. Iturriaga and S. Nesmachnow. Solving Very Large Optimization Problems (Up to One Billion Variables) with a Parallel Evolutionary Algorithm in CPU and GPU. In *Proceedings of the Sixth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pages 267–272, Victoria, Canada, 2012. ISBN 978-1-4673-2991-0.

S. Iturriaga, D. Garat, and G. Moncecchi. Restauración automática de acentos ortográficos en adverbios interrogativos. In *Proceedings of the XII Argentine Symposium on Artificial Intelligence (ASAI)*, JAIIO '11, pages 108–119, Córdoba City, Argentina, 2011.

S. Iturriaga, S. Nesmachnow, and B. Dorronsoro. A Multithreading Local Search For Multiobjective Energy-Aware Scheduling In Heterogeneous Computing Systems. In *Proceedings of the 26th European Conference on Modelling and Simulation (ECMS)*, pages 497–503, Koblenz, Germany, 2012a. ISBN 978-0-9564944-4-3.

S. Iturriaga, S. Nesmachnow, F. Luna, and E. Alba. A parallel online GPU scheduler for large heterogeneous computing systems. In *Proceedings of the 5th High-Performance Computing Latin America Symposium (HPCLatAm)*, JAIIO '12, Buenos Aires, Argentina, 2012b.

S. Iturriaga, S. Nesmachnow, and C. Tutté. Metaheuristics for multiobjective energy-aware scheduling in heterogeneous computing systems. In *EU/Metaheuristics Meeting Workshop (EU/ME)*, Copenhaguen, Denmark, 2012c.

S. Iturriaga, S. Nesmachnow, B. Dorronsoro, and P. Bouvry. Energy efficient scheduling in heterogeneous systems with a parallel multiobjective local search. *Computing and Informatics Journal (CAI)*, 2013a. Accepted on November 2012, to appear.

S. Iturriaga, S. Nesmachnow, F. Luna, and E. Alba. A parallel local search in CPU/GPU for scheduling independent tasks on large heterogeneous computing systems. *Journal of Parallel and Distributed Computing (JPDC)*, 2013b. Submitted on January 2013, pending acceptance.

S. Iturriaga, P. Ruiz, S. Nesmachnow, B. Dorronsoro, and P. Bouvry. A Parallel Multi-objective Local Search for AEDB Protocol Tuning. In *Proceedings of the 16th International Workshop on Nature Inspired Distributed Computing*, in the 27th IEEE/ACM International Parallel & Distributed Processing Symposium, Boston, Massachusetts, USA, 2013c. Accepted on February 2013, to appear.

H. Izakian, A. Abraham, and V. Snasel. Comparison of Heuristics for Scheduling Independent Tasks on Heterogeneous Distributed Environments. In *Proceedings of the International Joint Conference on Computational Sciences and Optimization*, volume 1 of *CSO '09*, pages 8–12, Washington, DC, USA, 2009a. IEEE Press. ISBN 978-0-7695-3605-7.

H. Izakian, A. Abraham, and V. Snášel. Metaheuristic Based Scheduling Meta-Tasks in Distributed Heterogeneous Computing Systems. *Sensors*, 9(7):5339–5350, 2009b.

B. Jones. An Overview of the EGEE Project. In C. Türker, M. Agosti, and H.-J. Schek, editors, *Peer-to-Peer, Grid, and Service-Orientation in Digital Library Architectures*, volume 3664 of *Lecture Notes in Computer Science*, pages 1–8. Springer-Verlag, 2005. ISBN 978-3-540-28711-7.

Y. Kessaci, M. Mezmaz, N. Melab, E. Talbi, and D. Tuyttens. Parallel Evolutionary Algorithms for Energy Aware Scheduling. In *Intelligent Decision Systems in Large-Scale Distributed Environments*. Springer-Verlag, 2011. ISBN 978-3-642-21270-3.

S. Khan and I. Ahmad. A Cooperative Game Theoretical Technique for Joint Optimization of Energy Consumption and Response Time in Computational Grids. *IEEE Transactions on Parallel and Distributed Systems*, 20:346–360, 2009.

J.-K. Kim, H. Siegel, A. Maciejewski, and R. Eigenmann. Dynamic Resource Management in Energy Constrained Heterogeneous Computing Systems Using Voltage Scaling. *IEEE Transactions on Parallel and Distributed Systems*, 19:1445–1457, 2008.

K. Kim, R. Buyya, and J. Kim. Power Aware Scheduling of Bag-of-Tasks Applications with Deadline Constraints on DVS-enabled Clusters. In *Proceedings of the 7th IEEE Press Symposium on Cluster Computing and the Grid*, pages 541–548, 2007.

H. Kimura, M. Sato, Y. Hotta, T. Boku, and D. Takahashi. Emprical study on Reducing Energy of Parallel Programs using Slack Reclamation by DVFS in a Power-scalable High Performance Cluster. In *Proceedings of the IEEE Press Conference on Cluster Computing*, pages 1–10, 2006.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

J. Knowles and D. Corne. On metrics for comparing nondominated sets. In *Proceedings of the Congress on Evolutionary Computation*, volume 1 of *CEC '02*, pages 711–716, 2002.

J. Knowles and D. Corne. Properties of an adaptive archiving algorithm for storing nondominated vectors. *IEEE Transactions on Evolutionary Computation*, 7(2):100–116, april 2003. ISSN 1089-778X.

J. D. Knowles and D. W. Corne. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation*, 8(2):149–172, 2000. ISSN 1063-6560.

S. Knust and P. Brucker. *Complex Scheduling.* Springer-Verlag, 2006. ISBN 3540295453.

J. Kołodziej, S. U. Khan, and F. Xhafa. Genetic Algorithms for Energy-Aware Scheduling in Computational Grids. In *Proceedings of the International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, 3PGCIC '11, pages 17–24, Washington, DC, USA, 2011. IEEE Press. ISBN 978-0-7695-4531-8.

J. Kołodziej, S. U. Khan, and A. Zomaya. A Taxonomy of Evolutionary Inspired Solutions for Energy Management in Green Computing: Problems and Resolution Methods. In J. Kołodziej, S. U. Khan, and T. Burczynski, editors, *Advances in Intelligent Modelling and Simulation*, volume 422 of *Studies in Computational Intelligence*, pages 215–233. Springer-Verlag, 2012. ISBN 978-3-642-30153-7.

B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms.* Springer-Verlag, 4th edition, 2007. ISBN 3540718435, 9783540718437.

J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection.* MIT Press, Cambridge, MA, USA, 1992. ISBN 0-262-11170-5.

P. Krömer, V. Snášel, J. Platoš, A. Abraham, and H. Izakian. Scheduling Independent Tasks on Heterogeneous Distributed Environments by Differential Evolution. In *Proceedings of the International Conference on Intelligent Networking and Collaborative Systems*, INCOS '09, pages 170–174. IEEE, 2009.

P. Krömer, J. Platoš, V. Snášel, and A. Abraham. An implementation of differential evolution for independent tasks scheduling on GPU. *Hybrid Artificial Intelligent Systems*, pages 372–379, 2011.

A. D. Kshemkalyani and M. Singhal. *Distributed Computing: Principles, Algorithms, and Systems.* Cambridge University Press, New York, NY, USA, 1st edition, 2008. ISBN 0521876346.

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

Y. Lam, J. Coutinho, and W. Luk. Integrated Hardware/Software Codesign for Heterogeneous Computing Systems. In *Proceedings of the 4th Southern Conference on Programmable Logic*, pages 217–220, 2008.

Y. Lee and A. Zomaya. Minimizing Energy Consumption for Precedence-Constrained Applications Using Dynamic Voltage Scaling. In *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pages 92–99, 2009.

Y. Lee and A. Zomaya. Energy Conscious Scheduling for Distributed Computing Systems under Different Operating Conditions. *IEEE Transactions on Parallel and Distributed Systems*, 22:1374–1381, 2011.

J. Lenstra, D. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990. ISSN 0025-5610.

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 IEEE Press Symposium on Cluster Computing and the Grid*, CCGrid 2004, pages 301–308, 2004.

Y. Li, Y. Liu, and D. Qian. A Heuristic Energy-aware Scheduling Algorithm for Heterogeneous Clusters. In *Proceedings of the 15th International Conference on Parallel and Distributed Systems*, pages 407–413, 2009.

S. Lin and B. W. Kernighan. An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research*, 21(2):498–516, 1973.

P. Lindberg, J. Leingang, D. Lysaker, S. Khan, and J. Li. Comparison and analysis of eight scheduling heuristics for the optimization of energy consumption and makespan in large-scale distributed systems. *The Journal of Supercomputing*, 59(1):323–360, 2012.

R. E. Lord, J. S. Kowalik, and S. P. Kumar. Solving Linear Algebraic Equations on an MIMD Computer. *Journal of the ACM (JACM)*, 30(1):103–117, 1983. ISSN 0004-5411.

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.

Y. Ma, B. Gong, R. Sugihara, and R. Gupta. Energy-efficient deadline scheduling for heterogeneous systems. *Journal of Parallel and Distributed Computing*, 72(12):1725–1740, 2012. ISSN 0743-7315.

M. Maheswaran, S. Ali, H. Siegal, D. Hensgen, and R. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Proceedings of the 8th Heterogeneous Computing Workshop*, HCW '99, pages 30–44, 1999.

S. Martello, F. Soumis, and P. Toth. Exact and approximation algorithms for makespan minimization on unrelated parallel machines. *Discrete Applied Mathematics*, 75(2):169–188, 1997. ISSN 0166-218X.

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.

M. Mezmaz, N. Melab, Y. Kessaci, Y. Lee, E. G. Talbi, A. Zomaya, and D. Tuyttens. A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. *Journal of Parallel and Distributed Computing*, 71(11):1497–1508, 2011.

P. Moscato. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Technical Report C3P Report 826, California Institute of Technology, 1989.

S. Nesmachnow and S. Iturriaga. Multiobjective Scheduling on Distributed Heterogeneous Computing and Grid Environments Using a Parallel Micro-CHC Evolutionary Algorithm. In *Proceedings of the 6th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pages 134–141, Barcelona, Spain, 2011.

S. Nesmachnow and S. Iturriaga. Multiobjective grid scheduling using a domain decomposition based parallel micro evolutionary algorithm. *International Journal of Grid and Utility Computing (IJGUC)*, 2013. Accepted on January 2012, to appear.

S. Nesmachnow, H. Cancela, and E. Alba. Heterogeneous computing scheduling with evolutionary algorithms. *Soft Computing*, 15:685–701, 2010. ISSN 1432-7643.

S. Nesmachnow, F. Luna, and E. Alba. Time Analysis of Standard Evolutionary Algorithms as Software Programs. In *Proceedings of the 11th International Conference on Intelligent Systems Design and Applications*, pages 271–276, Córdoba, Spain, 2011.

S. Nesmachnow, B. Dorronsoro, J. E. Pecero, and P. Bouvry. Energy-aware scheduling on multicore heterogeneous grid computing systems. *Submitted to Journal of Grid Computing*, 2012a.

S. Nesmachnow, F. Luna, and E. Alba. An Efficient Stochastic Local Search for Heterogeneous Computing Scheduling. *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops and PhD Forum*, 0:593–600, 2012b.

A.-C. Orgerie, L. Lefevre, and J.-P. Gelas. Save Watts in Your Grid: Green Strategies for Energy-Aware Framework in Large Scale Distributed Systems. In *Proceedings of the 14th IEEE Press Conference on Parallel and Distributed Systems*, ICPADS '08, pages 171–178, 2008.

I. H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63:511–623, 1996. ISSN 0254-5330.

P. Pardalos and H. Romeijn. *Handbook of Global Optimization*. Number 2 in Nonconvex Optimization and Its Applications. Springer-Verlag, 2002. ISBN 9781402006326.

J. Pecero, P. Bouvry, H. Huacuja, and S. Khan. A Multi-objective GRASP Algorithm for Joint Optimization of Energy Consumption and Schedule Length of Precedence-Constrained Applications. In *Proceedings of the 9th IEEE Press Conference on Dependable, Autonomic and Secure Computing (DASC)*, pages 510–517. IEEE, 2011.

M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer-Verlag, 3rd edition, 2008. ISBN 0387789340, 9780387789347.

F. Pinel, J. Pecero, P. Bouvry, and S. U. Khan. A two-phase heuristic for the scheduling of independent tasks on computational grids. In *Proceedings of the International Conference on High Performance Computing and Simulation (HPCS)*, pages 471–477, 2011.

F. Pinel, B. Dorronsoro, and P. Bouvry. Solving very large instances of the scheduling of independent tasks problem on the GPU. *Journal of Parallel and Distributed Computing*, 73(1):101–110, 2013.

A. M. Rahmani and M. A. Vahedi. A Novel Task Scheduling in Multiprocessor Systems with Genetic Algorithm by Using Elitism Stepping Method. *INFOCOMP Journal of Computer Science*, 7(2):58–64, 2008.

E. Rashedi, H. Nezamabadi-pour, and S. Saryazdi. GSA: A Gravitational Search Algorithm. *Information Sciences*, 179(13):2232–2248, 2009. ISSN 0020-0255. Special Section on High Order Fuzzy Sets.

I. Rechenberg. *Evolution strategy: optimization of technical systems according to principles of biological evolution*. Number 15 in Problemata. Frommann-Holzboog, Stuttgart-Bad Cannstatt, 1973.

G. Ritchie and J. Levine. A fast, effective local search for scheduling independent jobs in heterogeneous computing environments. In *Proceedings of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group*, pages 178–183, 2003.

N. B. Rizvandi, J. Taheri, A. Y. Zomaya, and Y. C. Lee. Linear Combinations of DVFS-Enabled Processor Frequencies to Modify the Energy-Aware Scheduling Algorithms. In *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, pages 388–397, 2010.

M. Safe, J. Carballido, I. Ponzoni, and N. Brignole. On Stopping Criteria for Genetic Algorithms. In A. Bazzan and S. Labidi, editors, *Advances in Artificial Intelligence*, volume 3171 of *Lecture Notes in Computer Science*, pages 405–413. Springer-Verlag, 2004. ISBN 978-3-540-23237-7.

M. Saito. A Variant of Mersenne Twister Suitable for Graphic Processors. Computing Research Repository (CoRR), 2010. URL `http://arxiv.org/abs/1005.4973`.

A. Salem, G. Anagnostopoulos, and G. Rabadi. A Branch-and-Bound Algorithm for Parallel Machine Scheduling Problems. In *Proceedings of the Harbour, Maritime and Multimodal Logistics Modeling and Simulation Workshop*, pages 88–93, Portofino, Italy, 2000. Society for Computer Simulation International (SCS).

J. Sanders and E. Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley, Inc., 1st edition, 2010. ISBN 0131387685, 9780131387683.

J. R. Schott. Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization. Master's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1995.

S. Seneviratne and D. C. Levy. Task profiling model for load profile prediction. *Future Generation Computer Systems*, 27(3):245–255, 2011. ISSN 0167-739X.

M. Sharifi, S. Shahrivari, and H. Salimi. PASTA: a power-aware solution to scheduling of precedence-constrained tasks on heterogeneous computing resources. *Computing*, 95:67–88, 2013. ISSN 0010-485X.

V. Shekar and B. Izadi. Energy aware scheduling for DAG structured applications on heterogeneous and DVS enabled processors. In *Proceedings of the International Green Computing Conference*, pages 495–502, 2010.

J. Shiers. The Worldwide LHC Computing Grid. *Computer Physics Communications*, 177(1–2):219–223, 2007. ISSN 0010-4655.

D. B. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62(3):461–474, 1993. ISSN 0025-5610.

G. Sih and E. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Transactions on Parallel and Distributed Systems*, 4(2):175–187, 1993. ISSN 1045-9219.

T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. De Micheli. Dynamic voltage scaling and power management for portable systems. In *Proceedings of the Design Automation Conference*, pages 524–529, 2001.

R. Storn and K. Price. Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11(4): 341–359, 1997. ISSN 0925-5001.

M. Studniarski. Stopping Criteria for Genetic Algorithms with Application to Multiobjective Optimization. In R. Schaefer, C. Cotta, J. Kołodziej, and G. Rudolph, editors, *Parallel Problem Solving from Nature, PPSN XI*, volume 6238 of *Lecture Notes in Computer Science*, pages 697–706. Springer-Verlag, 2010. ISBN 978-3-642-15843-8.

T. Stützle. *Local Search Algorithms for Combinatorial Problems: Analysis, Improvements, and New Applications*. Dissertationen zur Küntlichen Intelligenz. Infix, 1999. ISBN 9783896012203.

G. Subashini and M. Bhuvaneswari. Non Dominated Particle Swarm Optimizatio for Scheduling Independent Tasks on Heterogeneus Distributed Environments. *International Journal of Advances in Soft Computing and Its Applications*, 3(1), 2011a. ISSN 2074-8523.

G. Subashini and M. Bhuvaneswari. NSGA-II with Controlled Elitism for Scheduling Tasks in Heterogeneous Computing Systems. *International Journal of Open Problems in Computer Science and Mathematics (IJOPCM)*, 4(1), 2011b. ISSN 1998-6262.

H. Telega. Two-phase stochastic global optimization strategies. In *Foundations of Global Genetic Optimization*, volume 74 of *Studies in Computational Intelligence*, pages 153–197. Springer-Verlag, 2007. ISBN 978-3-540-73191-7.

H. Topcuoglu, S. Hariri, and M.-Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, 2002. ISSN 1045-9219.

B. Ucar, C. Aykanat, K. Kaya, and M. Ikinci. Task assignment in heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 66(1):32–46, 2006. ISSN 0743-7315.

S. Upadhyaya and S. Lata. Task allocation in distributed computing vs distributed database systems: A comparative study. *International Journal of Computer Science and Network Security*, 8(3):338–346, 2008.

S. L. van de Velde. Duality-Based Algorithms for Scheduling Unrelated Parallel Machines. *INFORMS Journal on Computing*, 5(2):192–205, 1993.

D. A. Van Veldhuizen. *Multiobjective evolutionary algorithms: classifications, analyses, and new innovations.* PhD thesis, Air Force Institute of Technology, 1999.

G. von Laszewski, L. Wang, A. Younge, and X. He. Power-aware scheduling of virtual machines in DVFS-enabled clusters. In *Proceedings of the IEEE Press Conference on Cluster Computing and Workshops*, pages 1–10, 2009.

C. Voudouris. Guided Local Search — an Illustrative Example in Function Optimisation. *BT Technology Journal*, 16(3):46–50, 1998. ISSN 1358-3948.

T. Wagner, H. Trautmann, and L. Martí. A Taxonomy of Online Stopping Criteria for Multi-Objective Evolutionary Algorithms. In R. Takahashi, K. Deb, E. Wanner, and S. Greco, editors, *Evolutionary Multi-Criterion Optimization*, volume 6576 of *Lecture Notes in Computer Science*, pages 16–30. Springer-Verlag, 2011. ISBN 978-3-642-19892-2.

N. Wilkins-Diehr, D. Gannon, G. Klimeck, S. Oster, and S. Pamidighantam. TeraGrid Science Gateways and Their Impact on Science. *Computer*, 41(11):32–41, 2008. ISSN 0018-9162.

M.-Y. Wu and D. Gajski. Hypertool: a programming aid for message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(3):330–343, 1990. ISSN 1045-9219.

M.-Y. Wu, W. Shu, and H. Zhang. Segmented Min-Min: A Static Mapping Algorithm for Meta-Tasks on Heterogeneous Computing Systems. In *Proceedings of the Heterogeneous Computing Workshop*, pages 375–385, 2000.

F. Xhafa. A Hybrid Evolutionary Heuristic for Job Scheduling on Computational Grids. In A. Abraham, C. Grosan, and H. Ishibuchi, editors, *Hybrid Evolutionary Algorithms*, volume 75 of *Studies in Computational Intelligence*, pages 269–311. Springer-Verlag, 2007. ISBN 978-3-540-73296-9.

F. Xhafa and A. Abraham. *Metaheuristics for Scheduling in Distributed Computing Environments.* Studies in Computational Intelligence. Springer-Verlag, 2008. ISBN 9783540692607.

F. Xhafa and A. Abraham. Computational models and heuristic methods for Grid scheduling problems. *Future Generation Computer Systems*, 26(4):608–621, 2010. ISSN 0167-739X.

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

F. Xhafa, J. Carretero, E. Alba, and B. Dorronsoro. Design and evaluation of tabu search method for job scheduling in distributed environments. In *Proceeding of the IEEE Press Symposium on Parallel and Distributed Processing*, IPDPS '08, pages 1–8. IEEE, 2008b.

B. Young, J. Apodaca, L. Briceño, J. Smith, S. Pasricha, A. Maciejewski, H. Siegel, B. Khemka, S. Bahirat, A. Ramirez, and Y. Zou. Deadline and energy constrained dynamic resource allocation in a heterogeneous computing environment. *The Journal of Supercomputing*, pages 1–22, 2012. ISSN 0920-8542.

Y. Yu and V. Prasanna. Power-aware resource allocation for independent tasks in heterogeneous real-time systems. In *Proceedings of the 9th International Conference on Parallel and Distributed Systems*, pages 341–348, 2002.

M. Zbigniew and D. Fogel. How to solve it: modern heuristics. *Springer-Verlag*, 2000.

L. M. Zhang, K. Li, and Y.-Q. Zhang. Green Task Scheduling Algorithms with Speeds Optimization on Heterogeneous Cloud Servers. In *Proceedings of the IEEE/ACM International Conference on Green Computing and Communications & International Conference on Cyber, Physical and Social Computing*, GREENCOM-CPSCOM '10, pages 76–80, Washington, DC, USA, 2010. IEEE Press. ISBN 978-0-7695-4331-4.

Y. Zhao, I. Raicu, and I. Foster. Scientific Workflow Systems for 21st Century, New Bottle or New Wine? In *IEEE Congress on Services - Part I*, pages 467–471, 2008. ISBN 978-0-7695-3286-8.

D. Zhu, R. Melhem, and B. Childers. Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multiprocessor Real-Time Systems. *IEEE Transactions on Parallel and Distributed Systems*, 14:686–700, 2003.

X. Zhu, C. He, Y. Bi, and D. Qiu. Towards Adaptive Power-Aware Scheduling for Real-Time Tasks on DVS-Enabled Heterogeneous Clusters. In *Proceedings of the IEEE/ACM International Conference on Green Computing and Communications & International Conference on Cyber, Physical and Social Computing*, GREENCOM-CPSCOM '10, pages 117–124, 2010.

X. Zhu, C. He, and J. Wang. Energy-efficient elastic scheduling in heterogeneous computing systems. In *Proceedings of the 30th IEEE Press Performance Computing and Communications Conference (IPCCC)*, pages 1–8, 2011.

E. Zitzler and L. Thiele. Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, PPSN V, pages 292–304, London, UK, UK, 1998. Springer-Verlag. ISBN 3-540-65078-4.

E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In K. C. Giannakoglou, D. T. Tsahalis, J. Périaux, K. D. Papailiou, and T. Fogarty, editors, *Proceedings of the Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems Conference*, EUROGEN '01, pages 95–100, Athens, Greece, 2001. International Center for Numerical Methods in Engineering (CIMNE).

E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003. ISSN 1089-778X.