Applied Soft Computing 12 (2012) 626–639

Contents lists available at SciVerse ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling

Sergio Nesmachnow[a,*], Héctor Cancela[a], Enrique Alba[b]

[a] Universidad de la República, Herrera y Reissig 565, Montevideo, Uruguay
[b] Universidad de Málaga, Spain

ARTICLE INFO

Article history:
Received 15 April 2011
Received in revised form 22 July 2011
Accepted 1 September 2011
Available online 4 November 2011

Keywords:
Parallel evolutionary algorithms
Scheduling
Heterogeneous computing
Grid

ABSTRACT

This work presents a novel parallel micro evolutionary algorithm for scheduling tasks in distributed heterogeneous computing and grid environments. The scheduling problem in heterogeneous environments is NP-hard, so a significant effort has been made in order to develop an efficient method to provide good schedules in reduced execution times. The parallel micro evolutionary algorithm is implemented using MALLBA, a general-purpose library for combinatorial optimization. Efficient numerical results are reported in the experimental analysis performed on both well-known problem instances and large instances that model medium-sized grid environments. The comparative study of traditional methods and evolutionary algorithms shows that the parallel micro evolutionary algorithm achieves a high problem solving efficacy, outperforming previous results already reported in the related literature, and also showing a good scalability behavior when facing high dimension problem instances.

1. Introduction

In the last 10 years, the fast increase of the processing power of low-cost computers and the rapid development of high-speed networking technologies have boosted the use of distributed computing environments to solve complex problems. Nowadays, a common platform for distributed computing usually comprises an heterogeneous collection of computers. The expression *grid computing* denotes the set of distributed computing techniques that work over a large loosely coupled virtual supercomputer, formed by combining together many heterogeneous platforms of different characteristics. This infrastructure has made it feasible to provide pervasive and cost-effective access to distributed computing resources for solving hard problems [16].

A key problem when using distributed heterogeneous computing (HC) environments consists in finding a scheduling strategy for a set of tasks to be executed. The goal is to assign the computing resources by satisfying some efficiency criteria, usually related to the total execution time or resource utilization. Scheduling problems on homogeneous multiprocessor systems have been widely studied in operational research [13,28]. However, the *heterogeneous computing scheduling problem* (HCSP) became specially important due to the popularization of distributed computing and the growing use of heterogeneous clusters since the 1990's [14,17].

Traditional scheduling problems are NP-hard [18], thus classic exact methods are only useful for solving problem instances of reduced size. Heuristics and metaheuristics are promising methods to solve the HCSP, since they are able to get efficient schedules in reasonable times, even for large problem instances. Evolutionary algorithms (EAs) have emerged as flexible and robust metaheuristic methods for solving the HCSP, achieving the high level of problem solving efficacy also shown in many other application areas [6]. EAs usually require larger execution times (in the order of a minute) than ad hoc scheduling heuristics, but they can find improved solutions. So, EAs are competitive schedulers for distributed HC and grid systems where large tasks (with execution times in the order of minutes, hours, and even days) are submitted for execution. In order to further improve the efficiency of EAs, parallel implementations have been employed to significantly enhance and speed up the search, allowing to reach high quality results in reasonable execution times even for hard-to-solve optimization problems [1].

EAs and other metaheuristics have been applied to the HCSP in the last 10 years. The proposals included genetic algorithms (GA) [9,36,41,45], memetic algorithms (MA) [39], cellular MA (cMA) [40], and hybrid approaches combining GA with other methods, such as neural networks [33], variable neighborhood search [37], and list scheduling techniques [11]. Two relevant works have obtained the best-known results when facing low-sized HCSP instances: an hybrid combining ant colony optimization (ACO) and Tabu search (TS) [32] that took over 3.5 h to perform the scheduling, and a hierarchic TS [42] that used a time limit of 100 s. Other HCSP variants

* Corresponding author.
E-mail address: sergion@fing.edu.uy (S. Nesmachnow).

1568-4946/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2011.09.022

have been faced using EAs, mostly remarkable the precedence-constrained task scheduling problem in multiprocessors [8,38], real-time grid scheduling [24], economy-based scheduling [44], and a complex HCSP version regarding many task attributes [10].

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

In our previous work [31], a parallel implementation of CHC (Cross generational elitist selection, Heterogeneous recombination, and Cataclysmic mutation) [15] was identified as a promising method for solving the HCSP. The parallel CHC achieved good solutions for a large set of HCSP scenarios, but it had two main drawbacks: the limited search pattern for structured scenarios and the loss of diversity that restrained it to work with no more than eight subpopulations.

In this line of work, the main contribution of this article is introducing parallel micro-CHC (pμ-CHC), a novel EA to solve the HCSP that combines a parallel subpopulations model with a focused evolutionary search using a micro population and a specific randomized local search (LS) method. Efficient numerical results are reported in the experimental analysis performed on both well-known problem instances and large HCSP instances. The comparative study of traditional methods and EAs shows that pμ-CHC is able to achieve high problem solving efficacy, outperforming previous results reported in the related literature, and also exhibiting good scalability when solving unseen high dimension problem instances.

The manuscript is structured as follows. The next section presents the problem formulation. Section 3 introduces EAs and describes the newly proposed pμ-CHC algorithm. Section 4 describes the implementation details of pμ-CHC applied to the HSCP. The discussion of the experimental analysis and results are presented in Section 5, while the conclusions and possible lines for future work are formulated in Section 6.

2. HCSP formulation

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

Scheduling problems mainly concern about time, trying to minimize the time spent to execute all tasks. In this model, the most usual metric to minimize is the *makespan*, defined as the time spent from the moment when the first task begins execution to the moment when the last task is completed [28]. The following formulation presents the mathematical model for the HCSP aimed at minimizing the makespan:

- given an HC system composed of a set of machines $P = \{m_1, m_2, \ldots, m_M\}$ (dimension M), and a collection of tasks $T = \{t_1, t_2, \ldots, t_N\}$ (dimension N) to be executed on the system,
- let there be an *execution time function* $ET: T \times P \rightarrow \mathbf{R}^+$, where $ET(t_i, m_j)$ is the time required to execute the task t_i in the machine m_j,
- the goal of the HCSP is to find an assignment of tasks to machines (a function $f: T^N \rightarrow P^M$) which minimizes the *makespan*, defined in Eq. (1).

$$\max_{m_j \in P} \sum_{t_i \in T : f(t_i) = m_j} ET(t_i, m_j) \tag{1}$$

In the presented HCSP formulation all tasks can be independently executed, disregarding the execution order. This kind of applications frequently appears in many lines of scientific research, and they are relevant in Single-Program Multiple-Data applications used for multimedia processing, data mining, parallel domain decomposition of numerical models for physical phenomena, etc. The independent tasks model also arises when different users submit their (obviously independent) tasks to execute in a computing service, specially in grid computing and volunteer-based computing infrastructures – such as TeraGrid, WLCG, Berkeley's BOINC, and Xgrid [7] – where non-dependent applications using domain decomposition are very often submitted for execution. Thus, the relevance of the HCSP version faced in this work is justified due to its significance in realistic distributed HC and grid environments.

The previous formulation defines the *static* HCSP. A static scheduler gathers all the available information about tasks and resources *before* the execution, and the task-to-resource assignment is not allowed to change during the execution. An accurate estimation of the execution time for each task on each machine is required by the scheduler, which is usually achieved by performing task profiling and statistical analysis of both submitted workloads and resource utilization. Static scheduling has it own areas of specific application, such as planning in distributed HC systems, and also analyzing the resource utilization for a given hardware infrastructure. Static scheduling also provides a first step for solving more complex scheduling problems arising in dynamic environments: an efficient static planner can be the building block to develop a powerful dynamic scheduler, able to deal with the increasing complexity of nowadays grid infrastructures.

Several deterministic heuristics have been proposed for HC scheduling [27]. Three of them have been used in this work to provide a baseline for comparing the results achieved with the new pμ-CHC algorithm:

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.

Sufferage identifies the task that if is not assigned to a certain host, it will *suffer* the most. The *sufferage value* is computed as the difference between the best MCT of the task and its second-best MCT, and this method gives precedence to those tasks with high sufferage value.

Min-Min greedily picks the task that can be completed the soonest. The method starts with a set U of all *unmapped* tasks, calculates the MCT for each task in U for each machine, and assigns the task with the minimum overall MCT to the machine that executes it faster. The mapped task is removed from U, and the process is repeated until all tasks are mapped. *Min-Min* improves upon the MCT heuristic, since it considers all the unmapped tasks sorted by MCT, and the availability status of the machines is updated by the least possible amount of time for every assignment. This procedure leads to more balanced schedules and it generally also allows finding smaller makespan values than other heuristics, since more tasks are expected to be assigned to the machines that can complete them the earliest.

3. Evolutionary algorithms

EAs are non-deterministic methods that emulate the evolutionary process of species in nature, in order to solve optimization, search, and learning problems [6,19]. In the last 25 years, EAs have been successfully applied for solving optimization problems underlying many real applications of high complexity. The proposed pμ-CHC method is a specialization of the CHC evolutionary algorithm originally proposed by Eshelman [15]. This section reviews the traditional CHC algorithm, presents concepts about parallelism and EAs, and introduces the novel pμ-CHC algorithm.

3.1. The CHC algorithm

CHC uses a conservative rank-based selection strategy that tends to perpetuate the best individuals in the population, and a special mating that only allows those individuals which differ from each other by some number of bits to reproduce. The initial threshold for allowing mating is often set to 1/4 of the chromosome length, and it is reduced by 1 each time that no offspring is inserted into the new population during the mating procedure. The recombination operator in CHC is Half Uniform Crossover (HUX), which randomly swaps exactly half of the bits that differ between the two parent encodings. A fitness-based replacement criterion is used: the new offspring must compete with their parents for survival. CHC does not apply mutation; diversity is provided by applying a reinitialization procedure, using the best individual found so far as a template for creating a new population after convergence is detected.

Algorithm 1 presents the pseudo-code for the CHC algorithm, showing those features that make it different from traditional EAs: the highly elitist replacement strategy, the use of HUX and reinitialization operators, and the mating restriction policy that does not allow to recombine a pair of "too similar" individuals.

Algorithm 1. Schema of the CHC algorithm.

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

3.2. Parallel evolutionary algorithms

Using multiple populations to improve the efficiency and the efficacy of EAs was proposed in the pioneering works by Mühlenbein [30] and Schlierkamp-Voosen and Mühlenbein [34]. By splitting the population into several computing elements, parallel evolutionary algorithms (PEAs) allow reaching high quality results in a reasonable execution time even for hard-to-solve optimization problems [1]. The pμ-CHC algorithm proposed in this work is categorized within the *distributed subpopulations* model [4]: the population is split in several subpopulations (*demes*). Each deme runs a serial EA, and the individuals are able to interact only with other individuals in the deme. An additional *migration* operator is defined: occasionally some individuals are exchanged among demes, introducing a new source of diversity in the EA.

Fig. 2 shows the generic schema for the migration procedure used in a distributed subpopulations PEA. Two conditions control the migration procedure: *sendmigrants* determines when the exchange of individuals takes place, and *recvmigrants* establishes whether a foreign set of individuals has to be received or not. *Migrants* denotes the set of individuals to exchange with some other deme, selected according to a given policy. The schema uses a *selection for migration* explicitly different to the *selection for reproduction*; they both return a selected group of individuals to perform the needed operation, but following potentially different policies. The *sendmigration* and *recvmigration* operators carry out the exchange of individuals among demes according to a connectivity graph defined over them, most usually a unidirectional ring.

Algorithm 2. Schema for migration in PEAs.

```
1:  if sendmigrants then
2:    migrants ← selection for migration(P(generation))
3:    sendmigration(migrants)
4:  end if
5:  if recvmigrants then
6:    immigrants ← recvmigration()
7:    P(generation) ← insert(immigrants, P(generation))
8:  end if
```

3.3. Parallel micro-CHC algorithm

By splitting the global population, PEAs allow achieving high computational efficiency due to the limited interaction and the reduced population size within each deme. However, EAs quickly lose diversity in the solutions when using small populations, and the search suffers a premature convergence effect, leading to a stagnation situation. The mating restriction technique and the reinitialization operator used in CHC are usually not powerful enough to provide the required diversity to avoid premature convergence in the parallel model when using very small populations (i.e. less than 10 individuals per deme). Many alternatives have been proposed in the related literature to overcome the loss of diversity on EAs. In the quest for designing a fast and accurate version of the CHC algorithm for solving the HCSP, able to achieve high quality results in a reduced execution time, concepts from the micro-genetic algorithm (μ-GA) by Coello and Pulido [12] were incorporated in this work in order to design a parallel micro-CHC algorithm.

Back in 2001, μ-GA [12] was a novel proposal of EA in the context of multiobjective optimization that followed previous works by Goldberg [20], Krishnakumar [26] and Knowles and Corne [25] to speed up the resolution of real-world problems. In 1989, theoretical studies by Goldberg hinted that an elitist GA is able to converge when using a population size of only three individuals [20]. Goldberg suggested a GA that uses a small population, which evolves until reaching nominal convergence (i.e. when all individuals in the population are similar to each other), and then, a *reinitialization* operator is applied to generate a new population, while keeping the best individuals from the previous cycle. The μ-GA uses two populations to store memory along the search: the main population used in any EA, and a secondary *elite* population to store the best solutions found in the search. The elite population allows keeping diversity at a low computational cost, by using the best individuals found so far to perform the population reinitialization after a certain (low) number of generations (two to five). The EA suggested by Goldberg and implemented in μ-GA has many concepts in

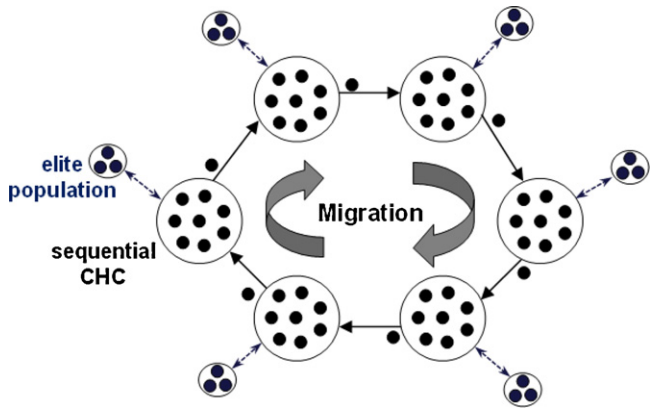


Fig. 1. Schema of the pμ-CHC algorithm.

common with the CHC method later proposed by Eshelman, thus those ideas can be directly applied to devise a micro-CHC algorithm.

The pμ-CHC algorithm introduced in this work combines a distributed subpopulation parallel model of the original CHC by Eshelman (using HUX and mating restriction) with two key concepts from μ-GA: an external population of elite solutions stored during the search, and an accelerated reinitialization using a randomized version of a well-known LS method to provide diversity within each subpopulation. A micro-population of eight individuals is used in each subpopulation of pμ-CHC. The size of the external population is three individuals, and a simple remove-of-the-worst strategy is used each time a new individual is inserted in the elite set. Fig. 1 shows a graphic schema of the distributed subpopulations model used in pμ-CHC. Additional features were included in order to design an efficient and fully scalable implementation for solving the HCSP (see Section 4 for the implementation details of pμ-CHC for the HCSP).

4. A parallel micro-CHC for the HCSP

This section presents the implementation details of the pμ-CHC algorithm, specifically developed in this work to solve the HCSP. The new algorithm was designed trying to achieve accurate solutions in a reduced time and to provide a good exploration pattern that allows scaling for solving large-size HCSP instances. To achieve these goals, several specific techniques have been applied in order to reduce the execution time and improve the evolutionary search.

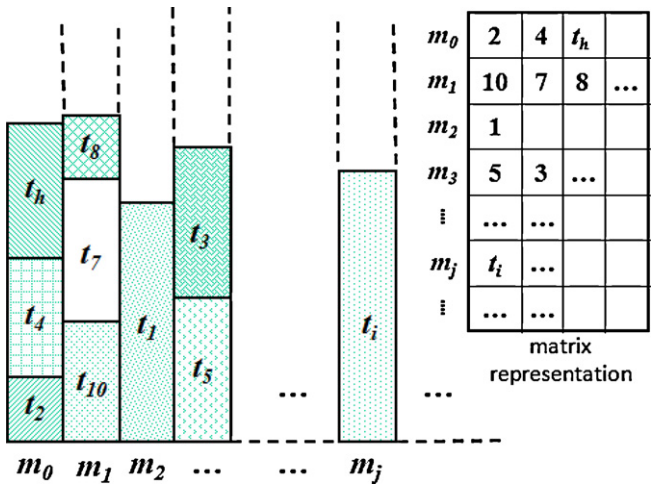


Fig. 2. Machine-oriented encoding.

4.1. The MALLBA library

MALLBA [2] is a library of optimization algorithms that can deal with parallelism in a user-friendly and, at the same time, efficient manner. The library implements several EAs as generic templates in *software skeletons* that will be instantiated with the problem features by the user. These templates incorporate the knowledge related to the resolution method, the interaction with the problem, and the parallelism. Skeletons are implemented by a set of *required* and *provided* C++classes that represent an abstraction of the entities participating in the resolution method:

- The *provided classes* implement internal aspects of the skeleton in a problem-independent way. The most important provided classes are `Solver` (the algorithm) and `SetUpParams` (for setting the algorithms' parameters).
- The *required classes* specify information related to the problem. Each skeleton includes the `Problem` and `Solution` required classes, that encapsulate the problem-dependent entities needed by the resolution method. Depending on the skeleton, other classes may be required.

The MALLBA library is publicly available to download at the University of Málaga website http://neo.lcc.uma.es/mallba/easy-mallba.

The implementation of pμ-CHC is based on the CHC skeleton provided by MALLBA. Additional code was incorporated into the skeleton to define and manage the external population, to implement the specialized reinitialization and local search operators used in pμ-CHC, and to include other features related to the HCSP resolution. The details about the problem encoding, the implementation of the evolutionary operators, and other specific features are provided in the next sections.

4.2. Problem encoding

pμ-CHC uses the *machine-oriented* encoding to represent HCSP solutions. The machine-oriented encoding uses a 2D structure in order to represent the group of tasks scheduled to execute on each machine m_j. Fig. 2 presents an example of the machine oriented encoding, showing for each machine m_j the list of tasks t_k assigned to it. The machine oriented encoding provides an easy and efficient way for performing exploration operators based on moving and swapping tasks, since it is able to store specific values of efficiency metrics for each machine (such as the local makespan). Thus, the makespan variation when performing changes on task assignments can be efficiently calculated considering only the tasks and machines involved in the move or swap performed, without requiring to reevaluate the efficiency metric for the whole schedule.

4.3. Initialization

Numerous methods have been proposed to initialize the population when applying EAs to the HCSP [9,40,43]. Usually, ad hoc scheduling heuristics have been used to start the evolutionary search from a set of useful suboptimal schedules, increasing the EA effectiveness to minimize the makespan. In pμ-CHC, the population is initialized using a randomized scheduling heuristic.

When dealing with low-dimension HCSP instances, deterministic heuristics provide easy-to-compute solutions to seed the population. *Min-Min* has been identified as an efficient scheduler for small HCSP instances [9], and also when the ET values has reasonable variations on heterogeneity [29], while *Sufferage* often achieves good schedules for non-structured scenarios. However, when the problem dimension grows, the time required to compute the solution using *Min-Min* and *Sufferage* increases, thus reducing the EA efficiency. To avoid the performance degradation,

the pμ-CHC population is initialized using probabilistic versions of *Min-Min* and *Sufferage*: they follow the general procedure of the deterministic heuristic, but only for assigning a random number of tasks, and the remaining tasks are assigned using a MCT strategy.

4.4. Variation operators

Exploitation: recombination. pμ-CHC uses HUX to recombine characteristics of two solutions. The list of tasks assigned to each machine in the offspring is chosen with uniform probability (0.5) between the ones in the parents. When a task is assigned to different machines in the parents, the machine to place that task in each offspring is also chosen with uniform probability.

Exploration: reinitialization. The reinitialization operator performs small perturbations in a given schedule, aimed at providing diversity to the population in order to avoid the search from getting stuck in local optima. It applies simple moves and swaps of tasks between two machines, selecting with high probability the machines with highest and lowest local makespan (`heavy` and `light`, respectively). The reinitialization is applied using the best individual found so far as a template for creating a new population after a stagnation situation is detected.

The reinitialization operator cyclically performs a maximum number of `MAXIT_REINIT` move-and-swap task operators, including: (1) move a randomly selected task (selecting the longest task with a probability of 0.5, and the rest with uniform probability) from `heavy` to `light`; (2) move the longest task from `heavy` to the suitable machine (the machine which executes that task in minimum time); (3) move into `light` the best task (the task with the lowest execution time for that machine); and (4) select a task from `heavy` (selecting the longest task with a probability of 0.5), then search the best machine to move it to.

Each time that a task is moved from a source machine to a destination machine, a swap between destination and source is randomly applied with a probability of 0.5. Unlike previous exploration operators for the HCSP [40,42], none of the foregoing operators imply exploring the $O(n^2)$ possible swaps, not even exploring the $O(n)$ possible task movements. The four exploration operators used in pμ-CHC are performed in sub-linear complexity order with respect to both the number of tasks and the number of machines in each HCSP instance. This feature allows pμ-CHC to show a good scalability behavior when solving large HCSP instances.

4.5. Local search: randomized PALS

Many strategies have been proposed in the related literature for providing diversity and improving the efficacy of the search when applying EAs to the HCSP. Most of the previous works concluded that LS methods are needed within any EA to find accurate schedules in short times. The works of Xhafa et al. [40–42] explored several LS operators for solving low-dimension HCSP instances, but many of the proposals become ineffective when the problem instances grow.

In our previous parallel CHC implementation [31], the reinitialization operator did not provide enough diversity when working with small subpopulations, thus limiting the parallel CHC (pCHC) to work using eight demes with 15 individuals each. In order to improve the population diversity, pμ-CHC incorporates a randomized version of Problem Aware Local Search (PALS), a novel heuristic algorithm originally proposed for the DNA fragment assembly problem [3].

PALS works on a single solution, which is iteratively modified by applying movements aimed to locally improve their function value. The key step is the calculation of the objective function variation when applying a certain movement. When it can be performed without significantly increasing the computational requirements, PALS provides an efficient search pattern for combinatorial optimization problems.

A specific variant of PALS was designed for the HCSP, aimed at exploring possible task swaps between machines in a given schedule, trying to improve the makespan metric. Due to the huge dimension of the search space, specially when solving large HCSP instances, the deterministic paradigm in PALS was replaced by a randomized one (i.e. the local search uses random criteria to define the set of adjacent swaps explored) in order to achieve accurate results in short execution times. The randomized version also incorporates two other differences with the generic PALS algorithm: (i) the main cycle ends when finding a solution that improves the schedule makespan, and (ii) when no improved solution is found, it performs `MAXIT_PALS` attempts applying the swap that produces the lowest makespan degradation, trying to introduce diversity in the population.

Algorithm 3 presents the randomized PALS for the HCSP. Working on a given schedule s, the method selects a machine m to perform the search. With high probability the machine with the largest local makespan is selected, focusing on improving the assignment for the machine which defines the makespan of the whole schedule, but also introducing a chance of improving the local makespan for other machines. The outer cycle iterates on `TOP_M` tasks assigned to machine m (randomly starting in task `start_m`), while the inner cycle iterates on `TOP_T` tasks assigned to other machines (randomly starting in task `start_t`). For each pair (t_M, t_T), the double cycle calculates the makespan variation when swapping tasks t_M and t_T. The method stores the best improvement on the makespan value for the whole schedule found in the `TOP_M` × `TOP_T` swaps evaluated. After the double cycle ends, the `best_move` found is applied, disregarding whether it produces an effective makespan reduction or not. The process is applied until finding a schedule which improves the original makespan or after performing `MAXIT_PALS` attempts.

Algorithm 3. Randomized PALS for the HSCP.

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

The randomized version of PALS was designed to provide an efficient and powerful search pattern for the HCSP. It allows moving towards local optima in the space of HCSP solutions each time that an improved solution is found, and it also provides diversity to solutions -allowing pμ-CHC to escape from strong local optima- after applying `MAXIT_PALS` changes when no improved solution is found. The calculation of the makespan variation when swapping two tasks (calculateDeltaMakespan(s, t_M, t_T)) is performed without

requiring high computational requirements, since the machine-oriented problem encoding stores the local makespan of each machine, thus PALS provides a very efficient search pattern for the HCSP. In pμ-CHC, the PALS operator was conceived to be applied to randomly selected individuals from the main and elite populations within a certain frequency.

4.6. Speeding up the pμ-CHC evolution

EAs quiclky lost diversity in the population when using low population sizes. In order to speed up the evolution of the pμ-CHC algorithm, the PALS operator is applied after a certain (low) number of generations pass without inserting any offspring into the new population during the mating procedure, following the idea of the traditional reinitialization operator in CHC. In preliminary experiments, this accelerated cataclysmic model was able to provide enough diversity to avoid pμ-CHC getting stuck in local optima. In this way, pμ-CHC combines the evolutionary search with PALS in order to achieve high accurate results in short execution times. In addition, a one-step memory is included: a task move is rejected when it will move the task back to the machine to which it was assigned one step in the past. The task memory is refreshed each time that a valid move is performed. This mechanism is a basic version of the memory already employed in TS algorithms to avoid loops, which has shown its usefulness for improving the HCSP results [42].

Summarizing, the distinctive characteristics of pμ-CHC includes:

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

4.7. Parallel model

A two-level parallel model was used in the implementation of pμ-CHC applied to the HCSP: the distributed memory message-passing paradigm was employed for communicating demes that execute in different hosts in a distributed cluster, while the shared-memory paradigm was applied in order to improve the efficiency of the communications between demes executing in the same host. This hybrid parallel implementation of pμ-CHC allows taking advantage of two types of parallel infrastructures: traditional clusters of computers, and also modern multicore CPU architectures, where several processing cores share a global memory that can be used to speed up the communications in cooperative-based distributed search methods. Both communication paradigms were implemented using the Message Passing Interface (MPI) [22], which is the most popular library used for developing parallel and distributed programs. By using the two-level parallel implementation, pμ-CHC diminishes the impact of the time spent in communication during the migration and synchronization procedures.

5. Experimental evaluation

This section introduces the set of HCSP instances used to evaluate the efficacy of the proposed EA. It also describes additional tools used to develop pμ-CHC, and the computational platform where the experimental evaluation was performed. After that, the experiments devoted to study the parameter settings of pμ-CHC are presented. The last section describes and analyzes the experimental results when solving both low-dimension and large-sized HCSP instances. It presents numerical results, a comparison with other techniques and lower bounds, and a statistical analysis on the improvements over the previous pCHC method. In addition, this section also includes a study of the makespan evolution and a scalability and parallel performance analysis when solving the large-dimension HCSP instances with pμ-CHC.

5.1. HCSP instances

Although the research community has faced the HCSP in the past, there do not exist standardized problem benchmarks or test suites for the problem [35]. When facing the HCSP, researchers have often used twelve instances proposed by Braun et al. [9], following the expected time to compute (ETC) performance estimation model by Ali et al. [5].

ETC takes into account three key properties: machine heterogeneity, task heterogeneity, and consistency. *Machine heterogeneity* evaluates the variation of execution times for a given task across the HC resources, while *task heterogeneity* represents the variation of the tasks execution times for a given machine. Regarding the consistency property, in a *consistent* 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. In an *inconsistent* scenario a given machine m_j may be faster than machine m_k when executing some tasks and slower for others. Finally, a *semi-consistent* scenario models those inconsistent systems that include a consistent subsystem.

All the HCSP instances from Braun et al. have 512 tasks and 16 machines, and they combine the three ETC properties in order to model several problem scenarios. The name of the instances has the pattern `d_c_MHTH.0`, where `d` stands for the distribution function used to generate the ETC values (`u`, for the uniform distribution), and `c` indicates the consistency type (`c` for consistent, `i` for inconsistent, and `s` for semiconsistent). `MH` and `TH` indicate the heterogeneity level for machines and tasks respectively (`lo` for low heterogeneity, and `hi` for high heterogeneity). Several HCSP suites were generated, but only the class `0` (the number after the dot) gained popularity.

For the purpose of studying the scalability of pμ-CHC as the problem instances grow, our experimental analysis will also consider a test suite of large-dimension HCSP instances, randomly generated to test the scalability and true limits of the proposed algorithm. This test suite of HCSP instances was designed following the methodology proposed by Ali et al. [5], in order to model large HC clusters and medium-sized grid infrastructures. It comprises HCSP instances with dimension (tasks $\times$ machines) 1024×32, 2048×64, 4096×128, and 8192×256. These dimensions are much larger than those of the popular benchmark by Braun et al. [9] and they better model present distributed HC and grid systems. For each dimension, twenty-four HCSP instances were generated regarding all the heterogeneity and consistency combinations, twelve of them considering the parametrization values from Ali et al. [5], and twelve using the values from Braun et al. [9]. The instances are named following the previously presented convention: the names have the pattern `M.d_c_MHTH`, where the first letter (`M`) describes the heterogeneity model (`A` for Ali, and `B` for Braun). The number `0` in the last position of the name was removed. The problem instances and the generator program are publicly available to download at http://www.fing.edu.uy/inco/grupos/cecal/hpc/HCSP.

Table 1
Makespan results of pμ-CHC for 512 × 16 HCSP instances from Braun et al. [9].

Instance	pCHC [31]			pμ-CHC			Imp. (avg)		Imp. (best)	
	Best	Avg.	σ	Best	Avg.	σ	Imp.	p-Value	Imp.	p-Value
u_c_hihi.0	7461819.1	7481194.5	0.26%	**7381570.0**	**7394702.7**	**0.09**%	1.16%	$<10^{-3}$	1.08%	$<10^{-3}$
u_c_hilo.0	153791.9	153924	0.06%	**153105.4**	**153193.7**	**0.04**%	0.47%	$<10^{-3}$	0.45%	0.009
u_c_lohi.0	241524.0	243446.3	0.29%	**239260.0**	**239706.2**	**0.08**%	1.54%	$<10^{-3}$	0.93%	$<10^{-3}$
u_c_lolo.0	5177.5	5181.6	0.07%	**5147.9**	**5152.3**	**0.04**%	0.57%	$<10^{-3}$	0.57%	$<10^{-3}$
u_i_hihi.0	2952493.2	2956905.7	0.21%	**2938380.8**	**2947896.4**	**0.14**%	0.30%	0.005	0.48%	0.003
u_i_hilo.0	73639.8	73847.1	0.13%	**73387.0**	**73531.4**	**0.10**%	0.43%	$<10^{-3}$	0.34%	0.009
u_i_lohi.0	102136.1	102677.3	0.30%	**102050.6**	**102402.8**	**0.17**%	0.27%	0.005	0.07%	0.01
u_i_lolo.0	2549.8	2557.2	0.11%	**2541.4**	**2547.1**	**0.09**%	0.39%	$<10^{-3}$	0.29%	$<10^{-3}$
u_s_hihi.0	4198779.5	4239146.3	0.36%	**4103500.3**	**4123537.3**	**0.27**%	2.73%	$<10^{-3}$	2.27%	$<10^{-3}$
u_s_hilo.0	96623.3	96750.3	0.13%	**95787.4**	**96020.5**	**0.10**%	0.75%	$<10^{-3}$	0.87%	$<10^{-3}$
u_s_lohi.0	123251.5	123989.4	0.24%	**122083.3**	**122744.4**	**0.23**%	1.00%	$<10^{-3}$	0.94%	$<10^{-3}$
u_s_lolo.0	3450.1	3472.2	0.13%	**3433.5**	**3438.3**	**0.07**%	0.98%	$<10^{-3}$	0.48%	$<10^{-3}$

5.2. Development and execution platform

pμ-CHC was implemented in C++, using the MALLBA library. The parallel implementation uses MPICH version 1.2.7p1, a well-known implementation of MPI [21], to perform the interprocess communications. Both the distributed memory (`ch_p4`) and the shared memory (`ch_shmem`) MPICH devices were employed to implement the two-level parallel model of pμ-CHC.

The experimental analysis was performed using a cluster with four Dell PowerEdge servers (QuadCore Xeon E5430 processors at 2.66 GHz, 8 GB RAM), using the CentOS Linux 5.2 operating system and Gigabit Ethernet LAN (cluster website: http://www.fing.edu.uy/cluster).

5.3. Parameter settings

The main objective of the research is to study the ability of the proposed pμ-CHC to *efficiently* solve the HCSP. Thus, a bounded effort stopping criterion fixed at 90 s of execution time is used, following the works by Xhafa et al. [40–42] and our previous pCHC implementation [31]. This time limit can be considered too long for scheduling short tasks in small multiprocessors, but it is actually an efficient time for scheduling in realistic distributed HC and grid infrastructures such as volunteer-computing platforms, distributed databases, etc., where large tasks – with execution times in the order of minutes, hours and even days – are submitted to execution. In addition, when facing large HCSP instances, ad hoc deterministic heuristics that usually require $O(n^3)$ operations also require execution times in the order of minutes (e.g. *Min-Min* needs more than a minute for computing schedules for HCSP instances with dimension 8192 × 256). The 90 s stopping criterion used in pμ-CHC is useful for efficiently solving static HCSP instances, and is also useful for solving dynamic scenarios following the rescheduling strategy, by replanning incoming and unexecuted tasks after certain intervals of time.

Regarding the crossover probability (p_C), the reinitialization probability (p_R), and the global population size (#*pop*), pμ-CHC used the best parameter values found in the previous work that evaluated a parallel CHC for the problem (p_C = 0.7, p_R = 0.8, #*pop* = 120) [31]. This decision allows performing a fair comparison between the two CHC proposals.

The parameter setting experiments studied the pμ-CHC subpopulation size, and parameters of the parallel model, using a subset of six problems from Braun et al., chosen to represent diverse HC scenarios. The migration operator in pμ-CHC considers the subpopulation connected in a unidirectional ring topology. The best results were achieved when using a micro population with eight individuals -thus, considering 16 demes-, and a selection for migration that exchanges the best two individuals between demes (the received individuals substitutes the worst ones in the destination deme). The best value for the migration frequency was 500 generations, providing a good balance between introducing diversity and reducing the time spent in communications.

The best results were obtained when applying the randomized PALS after five generations pass without inserting any offspring during the mating procedure. The value of `H_M` (probability of selecting the heavy machine) was fixed at 0.7. The value of `TOP_M` (number of tasks of the heavy machine to process) was fixed at $N/M = 32$, while the value of `TOP_T` (number of tasks examined to swap) was fixed at $N/20$. The number of iterations (`MAXIT_PALS`) was set to 7.

Summarizing, the parameter configuration that allowed achieving the best results in the configuration experiments was:

- *general parameters*: population size: 8 individuals, 16 demes, p_C = 0.7, p_R = 0.8.
- *migration*: selection: elitist, replacement: replace-worst, frequency: 500 generations.
- *randomized PALS*: application frequency: five generations without inserting any offspring, `MAXIT_PALS` = 7, `H_M`=0.7, `TOP_M` = N/M, `TOP_T` = $N/20$.

5.4. Results and discussion

This section discusses the experimental results of applying pμ-CHC to solve the HCSP.

5.4.1. Results for instances from Braun et al.

The results for the set of instances from Braun et al. are presented separately, since there are previous works that solved the benchmark using metaheuristic methods. Table 1 reports the best, average, and standard deviation on the makespan results obtained in 50 independent executions of pμ-CHC and the results obtained with our previous pCHC method [31]. The (percentaged) improvement factors on the makespan values achieved when using pμ-CHC over the previous pCHC are also reported. The (non-parametric) Kruskal–Wallis test was performed to analyze the results distributions, and the correspondent p-values are presented for each problem instance.

Table 1 shows that the new algorithmic proposal in pμ-CHC improves over the traditional parallel CHC method. By using a micro population, the accelerated evolution model, and the randomized PALS to improve the population diversity, pμ-CHC is able to find high-quality HCSP solutions with lower makespan values than the previously obtained with pCHC. Since the computed p-values are very small, the makespan improvements can be considered as statistically significant. The algorithmic robustness of pμ-CHC also

Table 2
Comparative makespan results: metaheuristics for 512 × 16 HCSP instances from Braun et al. [9].

Instance	GA [9]	MA+TS [39]	cMA [40]	ACO + TS [32]	TS [42]	pCHC [31]	pμ-CHC	t_B (s)
u_c_hihi.0	8050844.5	7530020.2	7700929.8	7497200.9	7448640.5	7461819.1	**7381570.0**	15
u_c_hilo.0	156249.2	153917.2	155334.8	154234.6	153263.3	153791.9	**153105.4**	62
u_c_lohi.0	258756.8	245288.9	251360.2	244097.3	241672.7	241524.0	**239260.0**	23
u_c_lolo.0	5272.3	5173.7	5218.2	5178.4	5155.0	5177.5	**5147.9**	50
u_i_hihi.0	3104762.5	3058474.9	3186664.7	2947754.1	2957854.1	2952493.2	**2938380.8**	44
u_i_hilo.0	75816.1	75108.5	75856.6	73776.2	73692.9	73639.8	**73387.0**	51
u_i_lohi.0	107500.7	105808.6	110620.8	102445.8	103865.7	102136.1	**102050.6**	77
u_i_lolo.0	2614.4	2596.6	2624.2	2553.5	2552.1	2549.8	**2541.4**	49
u_s_hihi.0	4566206	4321015.4	4424540.9	4162547.9	4168795.9	4198779.5	**4103500.3**	18
u_s_hilo.0	98519.4	97177.3	98283.7	96762	96180.9	96623.3	**95787.4**	46
u_s_lohi.0	130616.5	127633	130014.5	123922	123407.4	123251.5	**122083.3**	24
u_s_lolo.0	3583.4	3484.1	3522.1	3455.2	3450.5	3450.1	**3433.5**	30

improved with respect to pCHC, achieving very small values for the standard deviation on the makespan values. Therefore, it can be expected that pμ-CHC will find high-quality schedules in any single execution for small-sized HCSP scenarios that follow the ETC model by Ali et al. [5].

Table 2 presents the comparison of the pμ-CHC best makespan values against the best results previously found with diverse metaheuristic techniques. It also presents the time required by pμ-CHC to reach the (previous) best-known makespan value (t_B, in seconds). The analysis of Table 2 shows that pμ-CHC was able to compute better makespan values than the previous best-known solutions for all problem instances, outperforming the ACO+TS by Ritchie and Levine [32], the TS by Xhafa [42], and pCHC [31], the three previous best methods for solving the HCSP instances by Braun et al. In addition, short execution times were required to outperform the previous best-known results in all cases. The makespan values for the best solutions obtained are marked in bold. The best solutions (schedules) obtained for each problem instance are reported in the HCSP website http://www.fing.edu.uy/inco/cecal/hpc/HCSP.

After the previously presented results, we can claim that pμ-CHC is the new state-of-the-art algorithm for the small-sized HCSP instances by Braun et al. The next section characterizes its performance for solving unseen larger problems.

5.4.2. Results for new problem instances

This section presents the results for the new large HSCP instances designed to study the scalability of pμ-CHC when solving realistic distributed HC and grid scenarios. Tables 3–6 present the results for the large-sized HCSP instances. The tables report the best, average, and standard deviation on the makespan results achieved in 50 independent executions of pμ-CHC, and the results obtained with the *Min-Min* and *Sufferage* deterministic heuristics used as a reference baseline. The best results obtained with the previous pCHC method [31] is also presented in order to analyze the contribution of the new evolutionary model in pμ-CHC (using the micro-population, the external population, and the PALS operator to introduce diversity).

Tables 3–6 show that pμ-CHC computed significantly better results than *Min-Min* and *Sufferage* for all instances. pμ-CHC steadily improved over pCHC, while maintaining low values of standard deviation in the makespan, suggesting a robust behavior for large HCSP instances.

Table 7 summarizes the pμ-CHC improvement factors over the other studied scheduling methods. The makespan improvement factors were around 15% with respect to *Min-Min*, and around 20% (more than 25% for the largest instances) with respect to *Sufferage*. Lower improvement factors were obtained with respect to the

Table 3
Makespan results for HCSP instances of dimension 1024 × 32.

Instance	*Min-Min*	*Sufferage*	pCHC [31]	pμ-CHC		
				Best	Avg.	σ
A.u_c_hihi	22508064.0	26063096.0	20327924.0	**19676858.3**	19717711.4	0.11%
A.u_c_hilo	2255966.3	2694595.0	2048582.7	**1969980.0**	1975398.9	0.11%
A.u_c_lohi	2155.0	2537.5	1956.7	**1887.3**	1892.6	0.14%
A.u_c_lolo	225.9	261.0	207.5	**201.2**	201.5	0.08%
A.u_i_hihi	6367767.5	5601367.0	5169960.5	**5126273.0**	5147216.4	0.07%
A.u_i_hilo	641438.4	533545.2	490280.3	**485189.8**	487879.4	0.19%
A.u_i_lohi	664.7	551.7	518.2	**513.8**	516.8	0.24%
A.u_i_lolo	63.7	55.4	50.6	**50.2**	50.4	0.20%
A.u_s_hihi	14125880.0	14481880.0	12243560.0	**11837170.0**	11870719.2	0.15%
A.u_s_hilo	1319050.5	1379341.3	1187506.4	**1148940.6**	1155387.1	0.25%
A.u_s_lohi	1380.5	1417.8	1186.8	**1152.5**	1155.3	0.20%
A.u_s_lolo	138.7	141.0	122.4	**118.9**	119.4	0.18%
B.u_c_hihi	6708228.0	7874972.0	6169823.0	**6049220.5**	6052322.9	0.05%
B.u_c_hilo	66684.5	77250.5	61114.7	**59679.5**	59730.6	0.06%
B.u_c_lohi	232011.8	272422.6	215149.2	**210005.1**	210370.4	0.10%
B.u_c_lolo	2386.3	2826.9	2164.3	**2100.0**	2103.3	0.10%
B.u_i_hihi	2164576.5	1847652.5	1630288.6	**1616697.4**	1621628.0	0.11%
B.u_i_hilo	17083.1	16366.2	15121.5	**14993.2**	15047.8	0.26%
B.u_i_lohi	56601.2	55083.2	49569.9	**49060.5**	49351.9	0.31%
B.u_i_lolo	585.0	537.1	496.1	**487.5**	491.8	0.38%
B.u_s_hihi	3967265.5	3969449.5	3393010.2	**3255266.8**	3272088.3	0.22%
B.u_s_hilo	40691.6	41551.2	35988.4	**34675.2**	34747.2	0.19%
B.u_s_lohi	135624.6	132510.3	115179.2	**110749.7**	111068.5	0.20%
B.u_s_lolo	1333.2	1403.3	1191.7	**1153.1**	1158.0	0.24%

Table 4
Makespan results for HCSP instances of dimension 2048 × 64.

Instance	*Min-Min*	*Sufferage*	pCHC [31]	pμ-CHC		
				Best	Avg.	σ
A.u_c_hihi	19552222.0	25579850	18110479.1	**17474552.0**	17495744.9	0.11%
A.u_c_hilo	1873134.3	2478699.3	1748509.2	**1692750.0**	1699639.9	0.11%
A.u_c_lohi	1924.7	2539.2	1798.4	**1731.9**	1734.0	0.08%
A.u_c_lolo	191.7	249.8	177.6	**171.7**	171.8	0.05%
A.u_i_hihi	3248935.5	3218272.5	2506258.5	**2477753.9**	2492860.9	0.14%
A.u_i_hilo	365828.6	315267.5	272741.3	**272181.1**	272529.4	0.17%
A.u_i_lohi	320.9	312.5	266.3	**265.6**	266.2	0.20%
A.u_i_lolo	32.3	29.5	26.4	**26.3**	26.4	0.19%
A.u_s_hihi	11245679.0	13890956	9756499.7	**9359727.3**	9379560.0	0.13%
A.u_s_hilo	1042948.5	1307394.3	924094.9	**878838.4**	880125.4	0.13%
A.u_s_lohi	1056.0	1354.1	947.1	**911.8**	913.7	0.12%
A.u_s_lolo	114.6	142.3	99.6	**95.0**	95.1	0.09%
B.u_c_hihi	5564664.0	7560320.5	5290128.2	**5085005.2**	5092126.1	0.05%
B.u_c_hilo	59352.8	79079.2	55316.2	**53236.9**	53306.3	0.06%
B.u_c_lohi	190842.4	253468.1	177063.4	**170659.4**	170940.3	0.10%
B.u_c_lolo	1927.7	2613.8	1814.7	**1749.4**	1754.7	0.10%
B.u_i_hihi	929295.8	879421.3	770110.6	**763701.5**	766428.7	0.15%
B.u_i_hilo	10318.4	9047.6	7906.5	**7859.0**	7913.4	0.17%
B.u_i_lohi	34071.0	32073.6	26941.2	**26769.6**	26973.5	0.27%
B.u_i_lolo	355.7	299.4	262.4	**261.8**	263.2	0.26%
B.u_s_hihi	3293157.0	4121618.8	2910507.6	**2789531.9**	2796655.7	0.18%
B.u_s_hilo	33445.4	41777.5	29442.2	**28170.9**	28209.7	0.11%
B.u_s_lohi	111237.4	142534.7	98607.0	**93798.0**	93997.5	0.17%
B.u_s_lolo	1163.8	1474	1014.3	**969.7**	972.9	0.17%

previous pCHC method, but the differences increase as the problem instances grow, achieving a 3.11% of overall improvement factor for problem instances with dimension 8192 × 256. The figure in Table 7 presents a graphical summary of the improvements achieved by pμ-CHC with respect to the other scheduling methods.

Regarding the consistency classification, pμ-CHC obtained slight improvements over pCHC for inconsistent instances (around 1%), but the improvements were more significant in consistent (over 3%) and semi-consistent (over 4%) scenarios. These results show that pμ-CHC overcomes the problem of efficiently dealing with large structured scenarios detected for pCHC [31].

Table 8 and Fig. 3 summarize the averaged pμ-CHC improvements over the best makespan computed by *Min-Min* and *Sufferage* for each dimension and heterogeneity model. The makespan improvements over the best deterministic heuristic were always above **12%** for semiconsistent instances, and above **7%** for consistent instances. Lower improvement factors were obtained for small inconsistent instances, where *Min-Min* and *Sufferage* provide accurate solutions, but the improvements significantly increased up to more than **14%** for large inconsistent instances.

5.4.3. Makespan evolution and execution time

This section analyzes the variation of the best makespan results computed by pμ-CHC with respect to the wall-clock time. Fig. 4 presents the evolution of the ratio between the best makespan values obtained with pμ-CHC and the best deterministic

Table 5
Makespan results for HCSP instances of dimension 4096 × 128.

Instance	*Min-Min*	*Sufferage*	pCHC [31]	pμ-CHC		
				Best	Avg.	σ
A.u_c_hihi	16711134.0	23173816.0	15722681.0	**15260752.4**	15277595.2	0.05%
A.u_c_hilo	1649763.5	2240514.0	1562810.9	**1520225.1**	1521480.9	0.04%
A.u_c_lohi	1635.3	2248.6	1540.9	**1493.8**	1495.0	0.05%
A.u_c_lolo	166.9	223.9	155.7	**151.1**	151.2	0.05%
A.u_i_hihi	1666126.5	1575787.6	1309493.5	**1295054.0**	1312530.8	0.38%
A.u_i_hilo	177692.2	154506.9	137158.4	**135985.3**	137480.8	0.14%
A.u_i_lohi	188.0	165.6	136.1	**135.3**	136.6	0.19%
A.u_i_lolo	19.4	15.2	13.7	**13.6**	13.8	0.28%
A.u_s_hihi	8949853.0	11756833.0	8089853.5	**7831962.8**	7848970.9	0.13%
A.u_s_hilo	930564.0	1215532.5	828912.4	**799499.4**	801468.6	0.12%
A.u_s_lohi	927.9	1181.7	807.6	**778.3**	778.8	0.05%
A.u_s_lolo	94.7	122.3	84.2	**81.6**	81.7	0.13%
B.u_c_hihi	5059571.5	6912596.5	4767774.5	**4649566.5**	4651738.2	0.04%
B.u_c_hilo	49301.2	66003.5	46350.1	**45142.7**	45190.5	0.06%
B.u_c_lohi	169495.3	230424.2	158780.8	**154504.7**	154627.4	0.04%
B.u_c_lolo	1662.3	2263.6	1556.8	**1516.2**	1517.4	0.05%
B.u_i_hihi	524174.1	472071.9	402182.1	**398655.1**	403433.1	0.21%
B.u_i_hilo	5381.1	4964.7	4224.8	**4174.5**	4238.4	0.24%
B.u_i_lohi	18772.1	15873.5	13847.8	**13614.6**	13876.5	0.13%
B.u_i_lolo	183.9	152.4	137.4	**136.3**	137.9	0.31%
B.u_s_hihi	2843118.3	3551046.8	2508467.3	**2437604.5**	2440304.5	0.05%
B.u_s_hilo	27793.4	36605.5	25244.1	**24353.7**	24397.9	0.10%
B.u_s_lohi	91523.0	116056.8	81118.5	**78296.8**	78455.5	0.10%
B.u_s_lolo	921.8	1183.5	825.7	**800.7**	801.2	0.06%

Table 6
Makespan results for HCSP instances of dimension 8192 × 256.

Instance	*Min-Min*	*Sufferage*	pCHC [31]	pμ-CHC		
				Best	Avg.	σ
A.u_c_hihi	14798376.0	20198762.0	14070023.0	**13708686.6**	13712470.2	0.02%
A.u_c_hilo	1500181.5	2055377.3	1426068.0	**1388689.3**	1390076.8	0.04%
A.u_c_lohi	1456.5	2032.7	1384.8	**1344.2**	1344.8	0.03%
A.u_c_lolo	148.9	207.3	140.9	**136.6**	136.7	0.03%
A.u_i_hihi	878829.5	788940.8	702540.6	**690223.8**	693548.6	0.18%
A.u_i_hilo	85076.7	77317.0	70199.3	**68428.1**	70310.8	0.14%
A.u_i_lohi	96.1	82.6	71.0	**68.9**	71.4	0.76%
A.u_i_lolo	8.8	8.0	7.1	**6.9**	7.1	0.22%
A.u_s_hihi	8151522.0	10828664	7428847.5	**7112313.0**	7119414.4	0.06%
A.u_s_hilo	787507.6	1047018.1	711087.9	**685350.9**	686178.8	0.08%
A.u_s_lohi	796.9	1066.1	722.2	**691.5**	692.1	0.05%
A.u_s_lolo	81.2	107.9	73.8	**70.9**	71.0	0.05%
B.u_c_hihi	4460896.5	6251939.0	4254320.5	**4136265.2**	4137730.4	0.02%
B.u_c_hilo	43670.3	60967.2	41535.6	**40410.0**	40425.6	0.03%
B.u_c_lohi	148102.7	203203.7	140752.1	**136499.6**	136582.1	0.04%
B.u_c_lolo	1468.6	2000.6	1393.4	**1357.0**	1357.4	0.02%
B.u_i_hihi	286800.2	248651.3	211439.3	**205347.6**	212492.0	0.29%
B.u_i_hilo	2960.2	2496.7	2099.7	**2043.0**	2108.9	0.34%
B.u_i_lohi	9496.4	7887.3	7017.2	**6812.3**	7058.0	0.31%
B.u_i_lolo	90.0	78.8	71.0	**69.2**	71.5	0.31%
B.u_s_hihi	2411292.0	3137134.0	2155649.3	**2087688.3**	2088934.2	0.04%
B.u_s_hilo	23979.2	31826.8	21799.3	**21004.6**	21045.2	0.08%
B.u_s_lohi	79291.5	106247.6	72303.5	**69347.8**	69454.9	0.07%
B.u_s_lolo	807.2	1063.7	726.2	**696.3**	697.2	0.07%

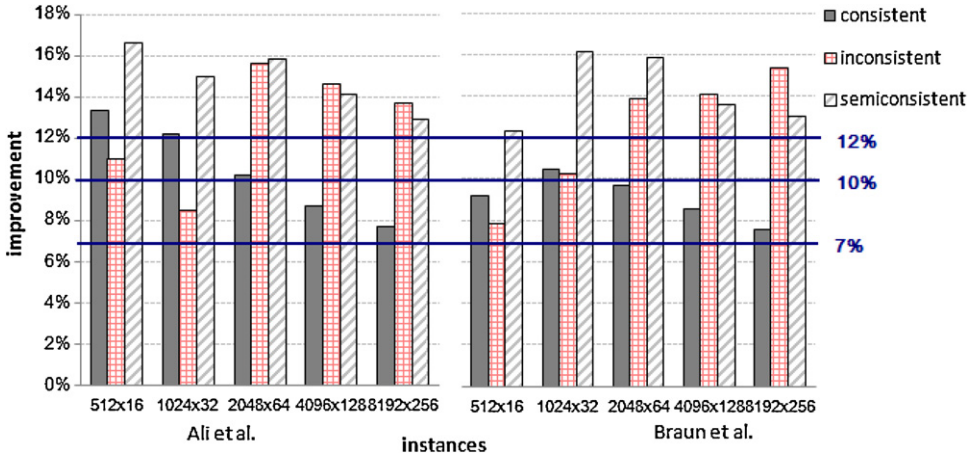


Fig. 3. pμ-CHC improvements over the best deterministic heuristic results, regarding the consistency classification.

heuristic, averaged for each problem dimension. The graphic shows that pμ-CHC was able to achieve high-quality results in low execution times for all HCSP instances. Less than 15 s are needed to achieve significant improvements with respect to the best deterministic heuristic in low-dimension problem instances (512 × 16 and 1024 × 32). The makespan reduction follows a more lethargic behavior for the largest problem instances (around a minute is needed to achieve 10% of makespan reduction).

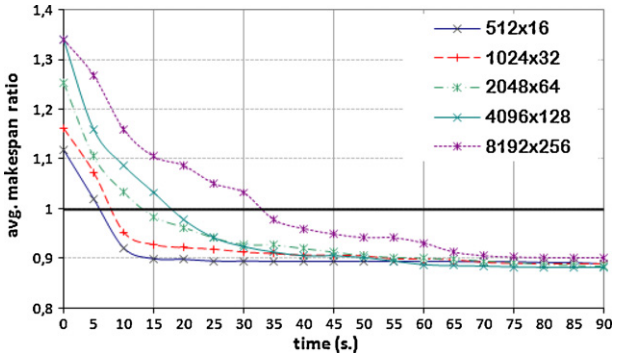


Fig. 4. Evolution of the makespan improvement ratio for pμ-CHC.

From a execution time-oriented point of view, Fig. 5 shows the time required to achieve a given improvement threshold in the makespan value with respect to the best deterministic heuristic result. pμ-CHC needs from 8 s (for dimension 512 × 16) to 45 s (for dimension 8192 × 256) to improve over the 5% threshold value, while 15 s (for dimension 512 × 16) to 80 s (for dimension 8192 × 256) of execution time are required to achieve an improvement of 10% over the best deterministic heuristic results.

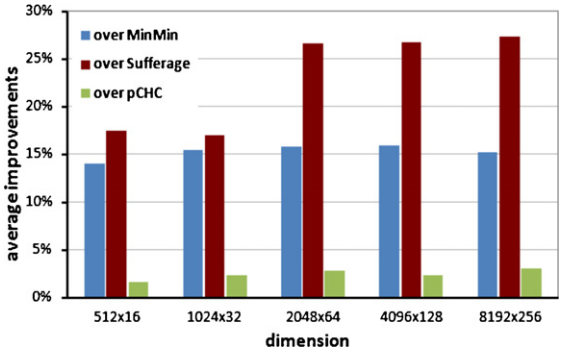
The previously presented results demonstrate the capacity of pμ-CHC to act as an efficient and accurate scheduler for heterogeneous computing and grid environments.

5.4.4. Comparison with lower bounds

Due to its high computational complexity, the non-preemptive HSCP cannot be solved in reasonable execution times using exact methods. However, a lower bound for the makespan value can be

Table 7
Overall average improvements when using pμ-CHC.

Model	Dimension	Improvement over		
		Min-Min	*Sufferage*	pCHC
Ali	512 × 16	15.32%	23.05%	2.48%
	1024 × 32	16.35%	16.92%	2.46%
	2048 × 64	15.76%	26.69%	2.75%
	4096 × 128	16.23%	27.16%	2.37%
	8192 × 256	14.46%	27.14%	3.11%
Braun	512 × 16	11.47%	13.55%	0.73%
	1024 × 32	14.57%	17.03%	2.41%
	2048 × 64	15.96%	26.66%	2.91%
	4096 × 128	15.68%	26.36%	2.31%
	8192 × 256	16.09%	27.56%	3.10%
Overall	512 × 16	14.10%	17.47%	1.61%
	1024 × 32	15.42%	16.98%	2.43%
	2048 × 64	15.86%	26.67%	2.83%
	4096 × 128	15.95%	26.76%	2.34%
	8192 × 256	15.27%	27.35%	3.11%



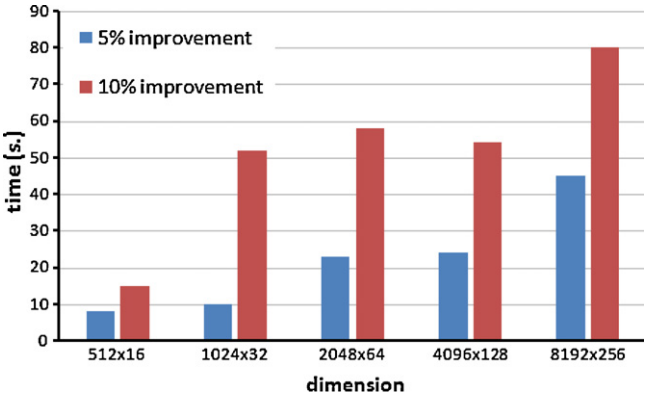


Fig. 5. Execution times required to achieve a given improvement threshold.

Table 9
Comparison with the lower bounds for the preemptive case.

Instance	LB	pμ-CHC (best)	GAP(LB)	Avg. GAP(LB)
u_c_hihi.0	7346524.2	7381570.0	**0.48%**	**0.38%**
u_c_hilo.0	152700.4	153105.4	**0.27%**	
u_c_lohi.0	238138.1	239260.0	**0.47%**	
u_c_lolo.0	5132.8	5147.9	**0.29%**	
u_i_hihi.0	2909326.6	2938380.8	**1.00%**	**0.73%**
u_i_hilo.0	73057.9	73387.0	**0.45%**	
u_i_lohi.0	101063.4	102050.6	**0.98%**	
u_i_lolo.0	2529.0	2541.4	**0.49%**	
u_s_hihi.0	4063563.7	4103500.3	**0.98%**	**0.82%**
u_s_hilo.0	95419.0	95787.4	**0.39%**	
u_s_lohi.0	120452.3	122083.3	**1.35%**	
u_s_lolo.0	3414.8	3433.5	**0.55%**	

Dimension	GAP(LB)		Imp.	% Ideal imp.
	Avg.	Best		
512 × 16	1.05%	0.75%	14.10%	94.93%
1024 × 32	2.32%	1.96%	15.42%	88.70%
2048 × 64	3.44%	3.15%	15.86%	82.18%
4096 × 128	4.70%	4.15%	15.95%	77.23%
8192 × 256	6.18%	5.07%	15.27%	71.18%

computed by solving the linear relaxation for the preemptive case of the problem. Under the preemption hypothesis the scheduler can temporarily interrupt a task and continue its execution on a different machine at a later time, without additional costs. In this (unrealistic) situation, an optimal solution has all machines with the same value of local makespan, which corresponds to the optimal makespan of the schedule.

The linear programming model relaxes the requirement that each task has to be assigned to one and only one machine, so the relaxed problem is to find a (normalized) vector of real numbers that represent the fraction of the task which is allocated to each machine. The linear programming problem is presented in Eq. (2).

$$\text{min} \quad \max_{j=1\ldots M} \sum_{i=1}^{i=N} x_{ij} \cdot ET(t_i, m_j)$$

$$\text{subject to} \quad \sum_{j=1}^{j=M} x_{ij} = 1; \quad 0 \leq x_{ij} \leq 1$$

The HCSP linear relaxation was solved with CPLEX [23], a software that uses the revised simplex method and the primal-dual interior point method to solve non-integer optimization problems such as the preemptive version of the HCSP. The lower bounds computed for the preemptive version are useful to determine the accuracy of the results achieved using the pμ-CHC method proposed in this work for solving the (non-preemptive) HCSP instances.

Table 9 summarizes the comparison between the pμ-CHC results and the lower bounds (LB) computed for the preemptive case, for each problem dimension. The results for the HCSP

Table 8
Improvements of pμ-CHC over the best deterministic heuristic.

Model	Type	Dimension				
		512 × 16	1024 × 32	2048 × 64	4096 × 128	8192 × 256
Ali	Consistent	13.34%	12.15%	10.18%	8.66%	7.68%
	Inconsistent	10.97%	8.44%	15.63%	14.64%	13.70%
	Semiconsistent	16.62%	14.97%	15.81%	14.14%	12.92%
Braun	Consistent	9.19%	10.45%	9.69%	8.54%	7.54%
	Inconsistent	7.77%	10.27%	13.85%	14.06%	15.35%
	Semiconsistent	12.28%	16.15%	15.86%	13.56%	13.03%

instances by Braun et al. are presented separately, since these scenarios are actively used by the research community. The lower bounds for the new instances are reported in the HCSP website.

Table 9 reports the relative gap value of the best and average results achieved for each problem dimension with respect to the correspondent lower bound (defined by Equation 2). The table also reports the improvements over the best deterministic heuristic, and the percentage of the ideal improvement achieved by pμ-CHC, considering the computed lower bounds.

$$GAP(LB) = \frac{result - LB}{LB} \quad (2)$$

The values reported in Table 9 show that pμ-CHC is able to compute accurate results when compared with the (in the general case, unattainable) lower bounds for the preemptive case. For the 512×16 HCSP instances the gaps were below 1.5% (and below 1% for 10 out 12 instances). The gaps increase as the problem dimension grows, but even for the largest problem dimension tackled in this work the gap values are below 6.20% (average) and 5.07% (best).

The previous results suggest that there is a small difference between the obtained results and the optimal makespan value for each problem instance (since the optimal makespan value lays between the computed LB and the pμ-CHC result). Fig. 6 presents a graphical summary of the improvements obtained by pμ-CHC over the best deterministic heuristic result and the comparison with the ideal improvement (given by the relative gap value with respect to the computed lower bounds), averaged for each problem dimension.

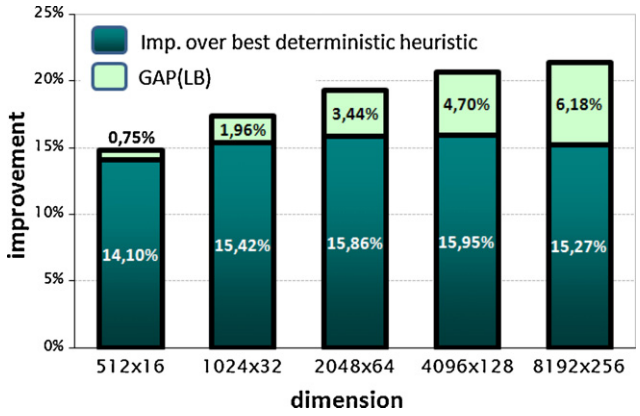


Fig. 6. pμ-CHC improvements with respect to the best deterministic heuristic results and gaps with the computed lower bounds.

5.4.5. Scalability analysis and parallel performance

The parameter setting experiments showed that pμ-CHC obtained the best makespan results for low-dimension HCSP instances when using the largest number of subpopulations considered (16). Additional experiments were performed evaluating the results of pμ-CHC when using different number of subpopulations to solve large-dimension problem instances. The pμ-CHC algorithm was executed using 2–16 subpopulations until reaching the time stopping criterion of 90 s, and the *normalized* makespan improvements (i.e. the ratio between the makespan achieved using n subpopulations and the makespan obtained with a single population) were evaluated.

Mean values of the average normalized makespan improvements achieved in 25 independent executions of pμ-CHC are reported in Fig. 7 for consistent, inconsistent and semiconsistent instances for each problem dimension studied. The graphic shows that the normalized makespan values diminish when solving large-dimension problem instances. These results demonstrate the ability of pμ-CHC of taking advantage of both the multiple evolutionary search and the randomized PALS operator in an efficient manner when using additional computational resources A sample cut of the 3D graphics for instances with a representative dimension (2048×64) and problem type (semiconsistent), is presented in Fig. 7, showing the reduction of the normalized makespan values (including error marks) when using additional subpopulations, in contrast with the behavior previously detected for the pCHC method, that suffered a degradation of the makespan results when more than eight demes were used [31]. Similar results were

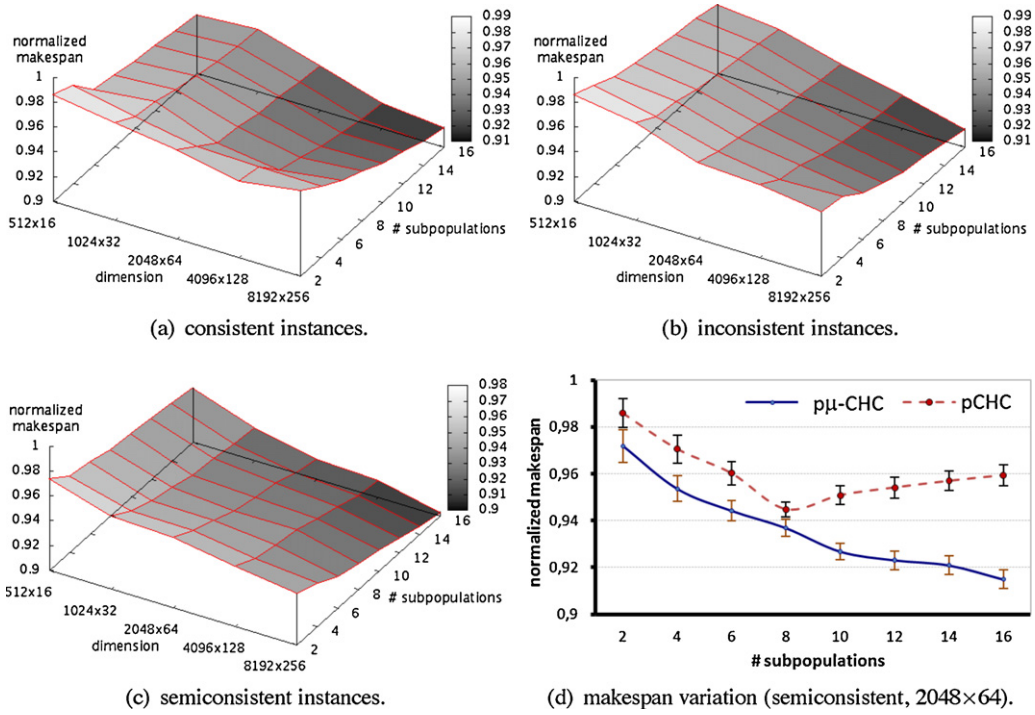


Fig. 7. Scalability and parallel performance analysis for pμ-CHC.

achieved when solving other HSCP instances. The new pμ-CHC is then a fully scalable scheduler, able to improve over its own results when using additional available computational resources.

6. Conclusions

This work presented a parallel micro-CHC applied to the HCSP, a crucial problem when executing tasks in HC systems. pμ-CHC was designed to efficiently solve large HCSP instances, using a bounded time stopping criterion that allows a quick planning. The new algorithm is inspired by multiobjective EAs, aimed at exploiting both the intrinsic parallel nature of EAs and the resource availability in grid environments. The pμ-CHC method uses a micro-population and follows an accelerated evolution model using a powerful randomized local search. It was implemented in MALLBA, using a two-level shared memory and message passing parallel model.

The experimental analysis solved benchmark HCSP instances and new high-dimension instances specially designed to analyze the scalability of the proposed method. The results demonstrate that pμ-CHC is an efficient scheduler for HC and grid environments, able to obtain good schedules in reduced execution times. pμ-CHC is the new state-of-the-art algorithm for the benchmark set of HCSP instances by Braun et al. [9], since it was able to improve over the previously best-known solutions computed with diverse metaheuristic techniques.

When solving high dimension HCSP instances up to 8192 tasks and 256 machines, pμ-CHC was also able to compute accurate results with respect to those obtained using traditional deterministic heuristics and a previous standard parallel implementation of CHC [31]. The makespan improvement factors obtained by pμ-CHC were **15%** over the *Min-Min* heuristic, **20–25%** over the *Sufferage* heuristic, and **3.5%** with respect to the previous pCHC method.

The scalability and parallel performance analysis showed that pμ-CHC is able to improve the makespan results when using more computing resources, and it also overcomes the problem of pCHC to deal with structured scenarios.

From the previous results, we can claim that pμ-CHC is a powerful tool for scheduling in HC and grid environments, when dealing with tasks having long execution times. In these scenarios, it is worth investing the time required for computing the schedule in order to achieve significant reductions (over **15%**) in the makespan values over deterministic heuristics.

The main lines for future work include to explore the role of each interacting element in pμ-CHC, and also to improve the algorithmic proposal in order to face still larger scenarios and the dynamic versions of the HCSP. Regarding the first issue, it would be interesting to evaluate the specific contribution of the pμ-CHC components, specially the local search operators, in order to get useful information about the fitness landscape to improve the results. On the other hand, new compact problem encodings and operators that perform larger modifications need to be devised to overcome the slow evolution pattern when solving large scenarios. Using different local search heuristics in each subpopulation is also an interesting extension of the proposed pμ-CHC algorithm. Future work could also focus on studying the applicability of pμ-CHC and other parallel EAs for solving dynamic versions of HCSP by using a rescheduling strategy. pμ-CHC is a powerful tool for quickly scheduling multiple tasks, therefore an iterated version of the algorithm could be used to perform rescheduling on dynamic scenarios. These lines of work are currently been investigated.

Acknowledgments

The work of S. Nesmachnow and H. Cancela has been partially supported by PEDECIBA and ANII, Uruguay. The work of E. Alba has been partially funded by the Spanish government and European FEDER (contract TIN2008-06491-C04-01, M* project), and by the Andalusian government (contract P07-TIC-03044, DIRICOM project).

References

[1] E. Alba, Parallel Metaheuristics: A New Class of Algorithms, Wiley-Interscience, 2005.
[2] E. Alba, F. Almeida, M. Blesa, C. Cotta, M. Diaz, I. Dorta, J. Gabarró, J. González, C. León, L. Moreno, J. Petit, J. Roda, A. Rojas, F. Xhafa, MALLBA: a library of skeletons for combinatorial optimisation, Parallel Comput. 32 (5-6) (2006) 415–440.
[3] E. Alba, G. Luque, A new local search algorithm for the DNA fragment assembly problem, in: Proc. of 7th European Conference on Evolutionary Computation in Combinatorial Optimization, vol. 4446 of Lecture Notes in Computer Science, Springer, 2007, pp. 1–12.
[4] E. Alba, M. Tomassini, Parallelism and evolutionary algorithms, IEEE Trans. Evol. Comput. 6 (5) (2002) 443–462.
[5] S. Ali, H. Siegel, M. Maheswaran, S. Ali, D. Hensgen, Task execution time modeling for heterogeneous computing systems, in: Proc. of the 9th Heterogeneous Computing Workshop, IEEE Press, Washington, USA, 2000, p. 185.
[6] T. Bäck, D. Fogel, Z. Michalewicz (Eds.), Handbook of Evolutionary Computation, Oxford University Press, 1997.
[7] F. Berman, G. Fox, A. Hey, Grid Computing: Making the Global Infrastructure a Reality, Wiley, New York, USA, 2003.
[8] W. Boyer, G. Hura, Non-evolutionary algorithm for scheduling dependent tasks in distributed heterogeneous computing environments, J. Parallel Distrib. Comput. 65 (9) (2005) 1035–1046.
[9] T. Braun, H. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, R. Freund, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, J. Parallel Distrib. Comput. 61 (6) (2001) 810–837.
[10] T. Braun, H. Siegel, A. Maciejewski, Y. Hong, Static resource allocation for heterogeneous computing environments with tasks having dependencies, priorities, deadlines, and multiple versions, J. Parallel Distrib. Comput. 68 (11) (2008) 1504–1516.
[11] P. Chitra, R. Rajaram, P. Venkatesh, Application and comparison of hybrid evolutionary multiobjective optimization algorithms for solving task scheduling problem on heterogeneous systems, Appl. Soft Comput. 11 (2) (2011) 2725–2734.
[12] C. Coello, G. Pulido, A micro-genetic algorithm for multiobjective optimization, in: Proc. of the First Int. Conf. on Evolutionary Multi-Criterion Optimization, Springer, London, UK, 2001, pp. 126–140.
[13] H. El-Rewini, T. Lewis, H. Ali, Task Scheduling in Parallel and Distributed Systems, Prentice-Hall, Inc., 1994.
[14] M. Eshaghian, Heterogeneous Computing, Artech House, 1996.
[15] L. Eshelman, The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination, in: Foundations of Genetics Algorithms, Morgan Kaufmann, 1991, pp. 265–283.
[16] I. Foster, C. Kesselman, The Grid: Blueprint for a Future Computing Infrastructure, Morgan Kaufmann, 1998.
[17] R. Freund, V. Sunderam, A. Gottlieb, K. Hwang, S. Sahni, Special issue on heterogeneous processing, J. Parallel Distrib. Comput. 21 (3) (1994).
[18] M. Garey, D. Johnson, Computers and Intractability, Freeman, 1979.
[19] D. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley, New York, 1989.
[20] D. Goldberg, Sizing populations for serial and parallel genetic algorithms, in: Proc. of the 3rd Int. Conf. on Genetic Algorithms, Morgan Kaufmann, San Francisco, USA, 1989, pp. 70–79.
[21] W. Gropp, E. Lusk, N. Doss, A. Skjellum, A high-performance, portable implementation of the MPI message passing interface standard, Parallel Comput. 22 (6) (1996) 789–828.
[22] W. Gropp, E. Lusk, A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, MIT Press, Cambridge, MA, 1994.
[23] ILOG, 2006. ILOG CPLEX 10.1: User's Manual, ILOG Inc., Mountain View, California, USA. http://www.gnu.org/software/glpk/glpk.html.
[24] G.V. Iordache, M.S. Boboila, F. Pop, C. Stratan, V. Cristea, A decentralized strategy for genetic scheduling in heterogeneous environments, Multiagent Grid Syst. 3 (4) (2007) 355–367.
[25] J. Knowles, D. Corne, Approximating the nondominated front using the pareto archived evolution strategy, Evol. Comput. 8 (2) (2000) 149–172.
[26] K. Krishnakumar, Micro genetic algorithms for stationary and nonstationary function optimization, in: Proc. of the SPIE Intelligent Control and Adaptive Systems Conference, 1989, pp. 289–296.
[27] Y. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Comput. Surv. 31 (4) (1999) 406–471.
[28] J. Leung, L. Kelly, J. Anderson, Handbook of Scheduling: Algorithms, Models, and Performance Analysis, CRC Press, Inc., 2004.
[29] P. Luo, K. Lü, Z. Shi, A revisit of fast greedy heuristics for mapping a class of independent tasks onto heterogeneous computing systems, J. Parallel Distrib. Comput. 67 (6) (2007) 695–714.
[30] H. Mühlenbein, Evolution in time and space - the parallel genetic algorithm, in: Foundations of Genetic Algorithms, Morgan Kaufmann, 1991, pp. 316–337.

[31] S. Nesmachnow, H. Cancela, E. Alba, Heterogeneous computing with evolutionary algorithms, Soft Comput. 15 (4) (2011) 685–699.
[32] G. Ritchie, J. Levine, A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments, in: Proc. of the 23rd Workshop of the UK Planning and Scheduling Special Interest Group, 2004, pp. 178–183.
[33] S. Salcedo-Sanz, Y. Xu, X. Yao, Hybrid meta-heuristics algorithms for task assignment in heterogeneous computing systems, Comput. Oper. Res. 33 (3) (2006) 820–835.
[34] D. Schlierkamp-Voosen, H. Mühlenbein, Strategy adaptation by competing subpopulations, in: Parallel Problem Solving from Nature, Springer-Verlag, 1994, pp. 199–208.
[35] M. Theys, T. Braun, H. Siegel, A. Maciejewski, Y. Kwok, Mapping tasks onto distributed heterogeneous computing systems using a genetic algorithm approach, in: Solutions to Parallel and Distributed Computing Problems, Wiley, New York, USA, 2001, pp. 135–178.
[36] L. Wang, H. Siegel, V. Roychowdhury, A. Maciejewski, Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach, J. Parallel Distrib. Comput. 47 (1) (1997) 8–22.
[37] Y. Wen, H. Xu, J. Yang, A heuristic-based hybrid genetic-variable neighborhood search algorithm for task scheduling in heterogeneous multiprocessor system, Inform. Sci. 181 (3) (2011) 567–581.
[38] A. Wu, H. Yu, S. Jin, K. Lin, G. Schiavone, An incremental genetic algorithm approach to multiprocessor scheduling, IEEE Trans. Parallel Distrib. Syst. 15 (9) (2004) 824–834.
[39] F. Xhafa, 2007. A hybrid evolutionary heuristic for job scheduling in computational grids, chap. 10, Springer Verlag Series: Studies in Computational Intelligence, vol. 75, pp. 269–311.
[40] F. Xhafa, E. Alba, B. Dorronsoro, Efficient batch job scheduling in grids using cellular memetic algorithms, in: Proc. 21st Int. Par. and Dist. Proc. Symposium, IEEE Press, Long Beach, California, USA, 2007, pp. 1–8.
[41] F. Xhafa, J. Carretero, A. Abraham, Genetic algorithm based schedulers for grid computing systems, Int. J. Innovative Comput. Inform. Control 3 (5) (2007) 1–19.
[42] F. Xhafa, J. Carretero, E. Alba, B. Dorronsoro, Design and evaluation of tabu search method for job scheduling in distributed environments, in: Proc. of the 22th Int. Par. and Dist. Proc. Symposium, IEEE Press, 2008, pp. 1–8.
[43] F. Xhafa, B. Duran, Parallel memetic algorithms for independent job scheduling in computational grids, in: Recent Advances in Evolutionary Computation for Combinatorial Optimization, vol. 153 of Studies in Computational Intelligence, Springer, 2008, pp. 219–239.
[44] J. Yu, R. Buyya, A budget constrained scheduling of workflow applications on utility grids using genetic algorithms, in: Proc. of the 15th IEEE Int. Symp. on High Performance Distributed Computing, IEEE Press, 2006, pp. 1–10.
[45] A. Zomaya, Y. Teh, Observations on using genetic algorithms for dynamic load-balancing, IEEE Trans. Parallel Distrib. Syst. 12 (9) (2001) 899–911.