Soft Comput (2011) 15:685–701
DOI 10.1007/s00500-010-0594-y

ORIGINAL PAPER

Heterogeneous computing scheduling with evolutionary algorithms

Sergio Nesmachnow · Héctor Cancela · Enrique Alba

Published online: 14 March 2010

Abstract This work presents sequential and parallel evolutionary algorithms (EAs) applied to the scheduling problem in heterogeneous computing environments, a NP-hard problem with capital relevance in distributed computing. These methods have been specifically designed to provide accurate and efficient solutions by using simple operators that allow them to be later extended for solving realistic problem instances arising in distributed heterogeneous computing (HC) and grid systems. The EAs were codified over MALLBA, a general-purpose library for combinatorial optimization. Efficient numerical results are reported in the experimental analysis performed on well-known problem instances. The comparative study of scheduling methods shows that the parallel versions of the implemented evolutionary algorithms are able to achieve high problem solving efficacy, outperforming traditional scheduling heuristics and also improving over previous results already reported in the related literature.

Keywords Evolutionary algorithms · Heterogeneous computing · Scheduling

S. Nesmachnow (✉) · H. Cancela
Universidad de la República, Montevideo, Uruguay
e-mail: sergion@fing.edu.uy

H. Cancela
e-mail: cancela@fing.edu.uy

E. Alba
Universidad de Málaga, Málaga, Spain
e-mail: eat@lcc.uma.es

1 Introduction

Distributed computing environments became popular in the past decades as a way to provide the needed computing power for solving complex problems. Usually, distributed computing environments are composed of many heterogeneous computers able to work cooperatively. At a higher level of abstraction, the expression *grid computing* denotes the set of distributed computing techniques that work over a large loosely coupled virtual supercomputer, formed by many heterogeneous components of different characteristics. This infrastructure has made it feasible to provide pervasive and cost-effective access to distributed computing resources for solving problems that demand large computing power (Foster and Kesselman 1998).

A key problem when using such heterogeneous computing (HC) systems consists in finding a planning strategy or *scheduling* for a set of tasks to be executed. The goal is to optimally 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, and numerous methods have been proposed for finding accurate schedules in reasonable times (El-Rewini et al. 1994; Leung et al. 2004). However, the *heterogeneous* computing scheduling problem (HCSP) has become important due the popularization of distributed computing and the growing use of heterogeneous clusters (Freund et al. 1994; Eshaghian 1996).

Traditional scheduling problems are NP-hard (Garey and Johnson 1979) and thus classic exact methods are only useful for solving problem instances of reduced size. Heuristics and metaheuristics are promising methods for

solving 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 (Bäck et al. 1997). Although they usually require larger execution times (in the order of a minute) than ad-hoc heuristics, EAs are able to find accurate solutions, so they are competitive schedulers for distributed HC and grid systems where large tasks (with execution times in the order of minutes, hours, and even days) are submitted to execution. In order to further improve the efficiency of EAs, parallel implementations have been employed to enhance and speed up the search, allowing to reach high-quality results in reasonable execution times even for hard-to-solve optimization problems (Alba 2005).

This work presents the application of sequential and parallel EAs for solving the HCSP. The study is aimed to analyze the efficacy of EAs to solve a de-facto standard set of (small-sized) problem instances, by using simple search operators that allow the methods to scale up in order to face realistic large problem instances. The main contributions of the work are to analyze EA techniques for solving the scheduling problem on distributed heterogeneous environments, to apply parallel models that allow improving the quality of results, and reducing the execution times, and to provide new state-of-the-art methods for HC scheduling that can be used as a building block for scaling to large distributed computing and grid systems.

The manuscript is structured as follows: The following section describes the paradigm of evolutionary computation and introduces the EAs involved in the study. Section 3 presents the HCSP formulation, some concepts about execution time estimation, and reviews previous works about applying EAs to solve the HCSP and related variants. Section 4 describes the implementation details of the sequential and parallel EAs used in the study. The experimental analysis and the discussion of the results are presented in Sect. 5, while the conclusions and possible lines for future work are formulated in Sect. 6.

2 Evolutionary algorithms

EAs are non-deterministic methods that emulate the evolutionary process of species in nature, in order to solve optimization, search, and other related problems (Davis 1991). In the past 25 years, EAs have been successfully applied for solving optimization problems underlying many real applications of high complexity.

The generic schema of an EA is shown in Algorithm 1. An EA is an iterative technique (each iteration is called a *generation*) that applies stochastic operators on a pool of individuals (the population P) in order to improve their *fitness*, a measure related to the objective function. Every individual in the population is the encoded version of a tentative solution of the problem. The initial population is generated by a random method or by using a specific heuristic for the problem. An evaluation function associates a fitness value to every individual, indicating its suitability to the problem. Iteratively, the probabilistic application of *variation operators* like the *recombination* of parts of two individuals or random changes (*mutations*) in their contents are guided by a selection-of-the-best technique to tentative solutions of higher quality.

The stopping criterion usually involves a fixed number of generations or execution time, a quality threshold on the best fitness value, or the detection of a stagnation situation. Specific policies are used to select the groups of individuals to recombine (the *selection* method) and to determine which new individuals are inserted in the population in each new generation (the *replacement* criterion). The EA returns the best solution ever found in the iterative process, taking into account the fitness function considered.

Algorithm 1 Schema of an evolutionary algorithm.

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

2.1 Genetic algorithm

The classic formulation of a GA can be found in Goldberg (1989). Based on the generic schema of an EA shown in Algorithm 1, a GA defines selection, recombination, and mutation operators, applying them to the population of potential solutions in each generation. In a classic application of a GA, the recombination operator is used to perform the search (exploiting the characteristics of suitable individuals), while the mutation is used as the operator aimed at providing diversity for exploring different zones of the search space. The simplest formulation (named *simple GA*) uses the Single Point Crossover (SPX)

as recombination operator and a mutation operator that randomly modifies selected positions in the solution encoding.

2.2 The CHC algorithm

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

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

Algorithm 2 Schema of the CHC algorithm.

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

2.3 Parallel evolutionary algorithms

Parallel implementations became popular in the last decade as an effort to improve the efficiency of EAs. By splitting the population into several processing elements, parallel evolutionary algorithms (PEAs) allow reaching high-quality results in a reasonable execution time even for hard-to-solve optimization problems (Alba 2005). The PEAs proposed in this work are categorized within the *distributed subpopulations* model according the classification by Alba and Tomassini (2002): the original population is divided into several subpopulations (*demes*), separated geographically from each other. Each deme runs a sequential EA, so individuals are able to interact only with other individuals in the deme. An additional *migration* operator is defined: occasionally some selected individuals are exchanged among demes, introducing a new source of diversity in the EA.

Algorithm 3 shows the generic schema for a distributed subpopulation PEA. It follows the generic schema of an EA, but including the pseudocode for the migration operator. 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 explicitly distinguishes between *selection for reproduction* and *selection for migration*; they both return a selected group of individuals to perform the operation, but following potentially different policies. The *sendmigration* and *recvmigration* operators carry out the exchange of individuals among demes according to a connectivity graph defined over them, most usually a unidirectional ring. Figure 1 presents a graphic representation of a distributed subpopulation PEA.

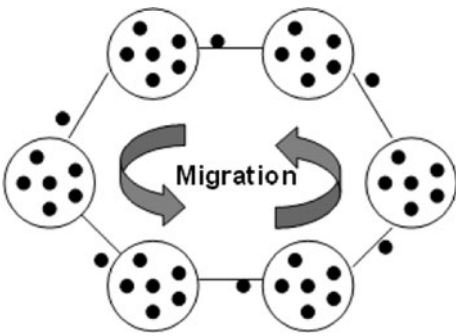

Fig. 1 Distributed subpopulations PEA

Algorithm 3 Schema of a parallel evolutionary algorithm.

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

3 Scheduling in heterogeneous environments

This section presents the HCSP mathematical formulation, some concepts regarding the task execution time estimation, and a review of previous works about applying EAs to solve the HCSP and related variants.

3.1 HCSP formulation

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

Scheduling problems mainly concern about time, trying to minimize the total time spent to execute all tasks. The most usual metric to minimize in this model 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. However, many other performance metrics have been considered in scheduling problems (Leung et al. 2004). 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 HC system,
- let there be an *execution time function* $\mathrm{ET} : P \times T \rightarrow \mathbf{R}^+$, where $\mathrm{ET}(t_i, m_j)$ 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.

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

The previous model does not account for task dependencies: all tasks can be independently executed, disregarding the execution order. Even though it is a simplified version of the more general scheduling problem that accounts for task dependencies, the independent task model is specially important in distributed environments such as supercomputing centers and grid infrastructures. Independent-task applications frequently appear in many lines of scientific research, and they are relevant in Single-Program Multiple-Data (SPMD) 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, such as Berkeley's BOINC, Xgrid, TeraGrid, EGEE, etc. (Berman et al. 2003), and in *parameter sweep applications*, structured as a set of multiple experiments, each one executed with a different set of parameter values. Thus, the HCSP version faced in this work is relevant due to its significance in realistic distributed HC and grid environments.

This work proposes using EAs to solve the *static* HCSP. A static scheduler gathers all the available information about tasks and resources *before* the execution. The scheduler takes early decisions, and the task-to-resource assignment is not allowed to change during the execution. Static schedulers require an accurate estimation of the execution time for each task on each machine, which is usually achieved by performing task profiling and statistical analysis of both submitted workloads and resource utilization. Static scheduling has its own areas of specific application, such as planning in distributed clusters and HC multiprocessors, and also analyzing the resource utilization for a fixed computing hardware. Static scheduling also provides a first step for solving more complex scheduling problems arising in dynamic environments: static results can be used as a reference baseline to determine if a dynamic scheduler is taking the right decisions on using the available resources in the system. In addition, an efficient static planner can be the building block to develop a powerful dynamic scheduler application capable of dealing with the increasing complexity of grid infrastructures of nowadays.

3.2 Execution time estimation

Execution time estimation is a common technique applied to model the execution time of tasks on a computer since the early 1990s (Yang et al. 1993). It relies on estimation methods such as task profiling, benchmarking, and statistical analysis, in order to provide an accurate prediction of the execution time of a given task on a specific machine. Researchers have stated that predicting the task execution times is useful to guide the resource selection performed by a scheduling method in HC environments (Li et al. 2004).

Ali et al. (2000) presented the *expected time to compute* (ETC) estimation model, which has been widely used in the past 9 years. ETC provides an estimation for the execution time of a collection of tasks in a HC system, taking into account three key properties: machine heterogeneity, task heterogeneity, and consistence. *Machine heterogeneity* evaluates the variation of execution times for a given task across the HC resources, while *task heterogeneity* represents the variation of the tasks execution times for a given machine.

The ETC model also considers a second classification. 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 kind of structured scenario captures the reality of many SPMD applications executing with local input data. An *inconsistent* ETC scenario lacks of structure among the computing requirements of tasks and the computing power of machines, so that a given machine m_j may be faster than machine m_k when executing some tasks and slower for others. This category represents a generic scenario for a distributed system composed of highly heterogeneous resources which receive many kinds of tasks. A *semi-consistent* ETC scenario models those inconsistent systems that include a consistent subsystem. There is not a predefined structure on the whole sets of tasks and machines in the HC system, but some of them behave like a consistent HC system.

The HCSP instances used to evaluate the EAs proposed in this work follow the ETC model by Ali et al. (see details in Sect. 5.1).

3.3 Related work: EAs for HC scheduling

This subsection presents a brief review of previous works that have proposed applying EAs to the HCSP and on related variants of this problem.

The early works on HC scheduling dates from 30 years ago, when the first ad-hoc deterministic heuristics for HC scheduling were proposed (Ibarra and Kim 1977; Davis and Jaffe 1981). The relevance of the HCSP increased in the 1990s due to the emergence of distributed computing, and the pioneering work on static HC scheduling in multiprocessor computers provided a foundation for solving more complex problem variants. Those seminal works provided the fist hints on the applicability of EAs to solve the HCSP and related problems.

Wang et al. (1997) created a line of research in HC scheduling, since many later works adopted their GA approach to solve diverse HCSP variants by using an ETC model. Using a seeding initialization to speed up the search, the GA by Wang et al. outperformed non-evolutionary heuristics for HCSP instances up to 100 tasks and 20 machines. The parallel GA by Kwok and Ahmad (1997) outperformed traditional scheduling methods for random graphs and numerical algorithms with up to 500 tasks in an Intel Paragon with 16 processors, while achieving almost linear speedup and good scalability. Grajcar (1999) combined a GA with a local search (LS) method for mapping an ordered set of tasks to an HC multiprocessor. The hybrid GA+LS found optimal solutions for instances with up to 96 tasks in short times, but it failed to find the optimum in large HC systems due to the lack of information about tasks and the environment (Grajcar 2001).

A new stage in applying EAs to the HCSP started in 2000, when Abraham et al. (2000) offered a conceptual description of applying nature-inspired methods to the HCSP. Later, Braun et al. (2001) presented a systematic comparison of 11 mapping heuristics for the HCSP, including a GA and an hybrid combining GA and Simulated Annealing. Both methods were able to obtain the best makespan values at that time for the HCSP scenarios studied, while they took benefit from using a seeded initialization to significantly improve the search. A load-balancing HCSP variant was faced by Zomaya and Teh (2001) using a centralized GA, which outperformed the First Fit heuristic and a random scheduler for instances up to 1,000 tasks and 50 processors. The load-balancing GA scheduler was effective from a practical point of view, especially when scheduling a large number of tasks.

Multiple works from Xhafa et al. have explored EAs applied to the HCSP. Duran and Xhafa (2006) studied a steady-state GA and the Struggle GA (SGA) for the HCSP. Both EAs outperformed the previous best makespan results in more than half of the instances studied, and an improved version of SGA using a task-resource allocation hash key obtained accurate results (Xhafa et al. 2008c). Xhafa et al. (2007) presented two GA schedulers, that achieved fast makespan reductions for HCSP instances up to 4,096 tasks and 256 machines. The Memetic Algorithm (MA) by Xhafa (2007) included subordinate LS methods to find

high-quality solutions in short times. The hybrid combining MA and Tabu Search (TS) achieved better results than previous GAs for the HCSP when using a 100 s stopping criterion. Two parallel models of MA+TS (Xhafa and Duran 2008) achieved several performance requirements on HCSP and grid scheduling. The structured population of a cellular MA (cMA) was used to control the tradeoff between the exploitation and exploration of the HCSP solution space (Xhafa et al. 2008a). Using a seeding initialization and three LS methods, cMA outperformed previous GA results for half the instances from Braun et al.

Other relevant works applied non-evolutionary methods to solve the HCSP. Ritchie and Levine (2004) proposed a hybrid combining Ant Colony Optimization (ACO) and TS for the HCSP. The hybrid ACO+TS outperformed previous makespan results for the HCSP instances by Braun et al., but it took a long time (over 3.5 h) to complete 1,000 iterations and thus it is not useful for scheduling in HC and grid environments. Xhafa et al. (2008b) presented a hierarchic TS that used a seeded initial solution, kept some elite solutions, and included several aspiration criteria. The elite solutions and the neighborhood structure were used to explore the search space, while several diversification methods were applied. Using a time stopping criterion of 90 s, TS improved over the previous best-known results in ten HCSP instances by Braun et al.

Many related variants of the HCSP have been faced using EAs. The GA by Wu et al. (2004) followed an *incremental* approach that gradually increases the difficulty of fitness values until finding an adequate schedule for precedence-constrained tasks in a multiprocessor system. However, the experimental analysis only considered four processors, and while the GA outperformed traditional scheduling methods in terms of solution quality, it needed large a population size, so the scalability of the approach to large problems is compromised. Boyer and Hura (2005) presented RS, an iterated randomized method for scheduling dependent tasks in HC systems, based in a ad-hoc scheduling heuristic that executes in linear order with respect to the number of tasks and machines. RS outperformed two GAs and a list scheduling (LSC) technique for simulated scenarios modeled by large ETC matrices and real world parallel applications, but failed to find the most efficient schedules in consistent scenarios when using a 100 s stopping criterion. The article did not present numerical results for each scenario; it only reported the comparative performance of RS against GAs and LSC. The recent work by Braun et al. (2008) tackled a general HCSP formulation regarding dependencies, priorities, deadlines, and versions. The experimental analysis compared two greedy heuristics, a GA, and a steady-state GA with customized encodings and operators to solve scenarios with up to eight machines and 2,000 tasks. The results showed that the steady-state GA performed the best among the studied heuristics, achieving the best schedules, whose fitness values were between 65 and 95% of upper bounds calculated under unrealistic assumptions.

The analysis of the related works shows the large diversity of proposals on applying EAs and other metaheuristics for solving the HCSP and related variants. Despite the existence of these numerous proposals, there have been few works studying 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 this line of research, by studying highly efficient parallel EA models that use simple operators, since this kind of methods could provide accurate solutions, even for large-sized hard-to-solve HCSP instances.

4 Designing evolutionary algorithms for the HCSP

This section presents two EAs applied to the HCSP: a traditional GA, and a CHC algorithm in their sequential and parallel variants. Both methods were designed aiming at achieving accurate solutions in reduced time, while providing a good exploration pattern, by using seeding initialization and special evolutionary operators. The details are presented in the next subsections, along with the software library in which the EAs were implemented.

4.1 Problem encoding

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

The task-oriented encoding uses a vector of machine identifiers to represent the task-to-resource assignment, as it is presented in Fig. 2. The one-dimension vector has N elements, where the presence of m_j in the position t_i means

Fig. 2 Task oriented encoding

that the task t_i is scheduled to execute on machine m_j. The task-oriented encoding is a direct representation for schedules that allows applying simple exploration operators based on moving and swapping tasks. However, after applying a move or swap operator, the task-oriented encoding does not provide an easy way to evaluate the changes on efficiency metrics of the whole schedule (such as makespan, flowtime or resource utilization). When using the task-oriented encoding, any single change on a task assignment forces to reevaluate the schedule metric.

The machine-oriented encoding uses a 2D structure in order to represent the group of tasks scheduled to execute on each machine m_j. Figure 3 presents an example of the machine-oriented encoding, showing for each machine m_j the list of tasks t_k assigned to it. The machine-oriented encoding provides an 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). So, any single change on a task assignment does not imply reevaluating the schedule metric.

Both encodings were studied in the EAs proposed in this work. The prototype implementations of traditional GA and CHC methods used the task-oriented encoding, in order to provide a simple method for analyzing the efficiency of EAs for solving the HCSP. In a second stage, the machine-oriented encoding was adopted, trying to improve the efficiency of the makespan calculation. Those improved GA and CHC versions allow to store the local makespan values, significantly enhancing the computational efficiency of the search. The parallel GA and CHC versions use the machine-oriented encoding in order to further improve the computational efficiency of the search.

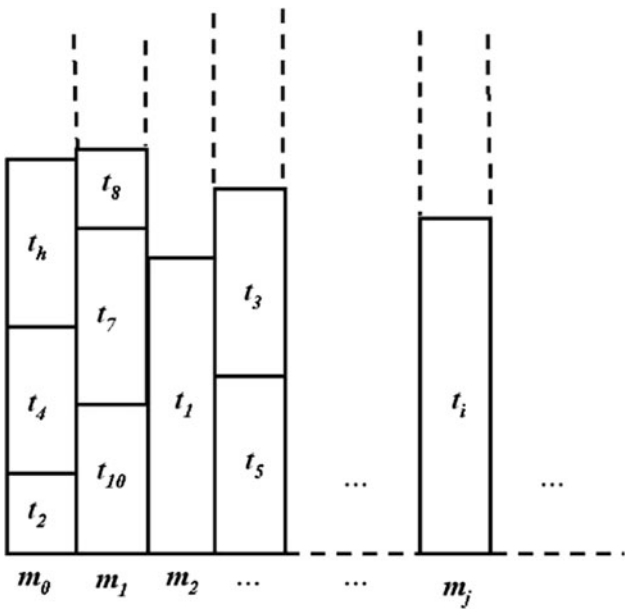

Fig. 3 Machine oriented encoding

4.2 The MALLBA library

MALLBA (Alba et al. 2002) is a library of algorithms for optimization that can deal with parallelism (on a Local Area Network (LAN) or on a Wide Area Network (WAN)), in a user-friendly and, at the same time, efficient manner. The EAs described in this section are implemented as generic templates on the library as *software skeletons*, to be instantiated with the features of the problem by the user. They incorporate all the knowledge related to the resolution method, its interactions with the problem, and the parallel considerations. 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 solver in a problem-independent way. The most important *provided* classes are `Solver` (the algorithm) and `SetUpParams` (setup parameters).
- The **required classes** specify information specifically related to the problem. Each solver includes the required classes `Problem` and `Solution`, that encapsulate the problem-dependent entities needed by the resolution method. Depending on the algorithm, other classes may be required.

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

4.3 Implementation details of the proposed EAs

This subsection presents the implementation details of the studied methods, describing the initialization, recombination, mutation, and reinitialization used in GA and CHC sequential, and parallel versions.

4.3.1 Initialization

Numerous methods have been proposed to generate the population in the related works on applying EAs to the HCSP. Many of them employed specific heuristics to start the evolutionary search from a set of useful suboptimal schedules. The EAs proposed in this work use the strategy of seeding the initial population using a randomized version of the Min–Min heuristic, following previous approaches by Braun et al. (2001), Xhafa et al. (2007), and others.

Min–Min starts with a set U of all *unmapped* tasks, calculates the minimum completion time (MCT) for each task in U for each machine, and assigns the task with the minimum overall MCT to the best machine. The mapped task is removed from U, and the process is repeated until all tasks are mapped. Min–Min considers the unmapped tasks sorted by MCT, and the availability status of the machines are updated by the least possible amount of time for every assignment. This procedure leads to balanced schedules and generally also allows finding smaller makespan values than other heuristics, since more tasks are expected to be assigned to the machines that complete them the earliest. The Min–Min strategy has been identified as an efficient method for finding accurate schedules for small size HCSP instances (Braun et al. 2001), and also for ETC matrices with reasonable heterogeneity variations (Luo et al. 2007).

The initialization operator used in the proposed EAs follows the general procedure of the Min–Min heuristic, but only for assigning a random number of tasks, while the remaining tasks are assigned using a load balancing strategy.

4.3.2 Exploitation: recombination

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

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

4.3.3 Exploration: mutation and reinitialization

Both the mutation (GA) and reinitialization (CHC) are random operators that perform small perturbations in a given schedule, aimed at providing diversity to the population, to avoid the search to get stuck in local optima. These operators explore simple moves and swaps of tasks between two machines, selecting with high probability the machines with highest and lowest local makespan (`heavy` and `light`, respectively). The GA applies the mutation operator on selected individuals along the evolutionary search, while CHC applies the reinitialization using the best individual found so far as a template for creating a new population after convergence is detected.

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

- Move a randomly selected task (selecting the longest task with a probability of 0.5) from `heavy` to `light`.
- Move the longest task from `heavy` to the suitable machine (the machine which executes that task in minimum time).
- Move into `light` the best task (the task with the lowest execution time for that machine).
- 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 presented in related works by Xhafa et al. Xhafa et al. (2006, 2007), none of the foregoing operators imply exploring the $O(n^2)$ possible swaps, not even exploring the $O(n)$ possible task movements. The exploration operators used in the proposed EAs are all performed in sub-linear complexity order with respect to both the number of tasks and the number of machines in each HCSP instance. This feature allows the EAs to have a good scalability behavior when applied to solve large HCSP and grid scheduling problem instances.

The pseudo-code of the mutation and reinitialization operators for the HCSP is presented in Algorithm 4. In order to further improve the schedules, a rebalancing operator is randomly applied with probability 0.5 after performing the mutation operator. This operator was designed in order to fill the gap on local makespan between the heavy machine and the light machine, by performing task moves between them while it is possible, trying to balance the load in the schedule.

Algorithm 4 HCSP mutation and reinitialization.

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

5 Experimental analysis

This section introduces the set of HCSP instances and the computational platform used for the experimental evaluation of the proposed EAs. After that, the parameter setting experiments for the algorithms are commented. The last subsection presents and analyzes the experimental results for sequential and parallel versions of the applied EAs. It presents numerical results, a comparison with other techniques and lower bounds, and a statistical analysis on the results improvements. In addition, the analysis of the makespan evolution and the execution time is presented for representative executions of parallel CHC. Finally, a scalability analysis studies the behavior of parallel CHC when solving large-sized HCSP instances.

5.1 HCSP instances

Although the research community has faced the HCSP in the past, there has been little effort to define standardized problem benchmarks or test suites. Braun et al. (2001) presented a test suite of 12 instances generated using the ETC model. All the instances have 512 tasks and 16 machines, and they combine the three ETC model properties (task and machine heterogeneity, and consistency) in order to model several problem scenarios. The 12 instances by Braun et al. have become a de facto standard benchmark suite for evaluating algorithms for solving the HCSP.

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

5.2 Development and execution platform

The EAs were implemented in C++, using the MALLBA library. The experimental analysis was performed on a cluster of 4 Dell PowerEdge servers, with QuadCore Xeon E5430 processors at 2.66 GHz, 8GB RAM, connected with a Gigabit Ethernet LAN and the CentOS Linux 5.2 operating system (cluster website: http://www.fing.edu.uy/cluster).

5.3 Parameter settings

The EAs studied in this work used a 90-s execution time stopping criterion, following the previous works by Xhafa et al. (2007, 2008a, b). This time limit can be considered too high for scheduling short tasks in small multiprocessors, but it is 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.

Instead of fixing an arbitrary set of parameters, an initial configuration analysis was performed for determining the best parameter values for each operator in the sequential GA and CHC variants. The parameter setting analysis was performed using a subset of three problems with diverse

characteristics (`u_i_hilo.0`, `u_c_hihi.0`, and `u_s_lohi.0`). The studied parameters included population size, crossover probability (p_C), mutation probability (p_M) in GA, and percentage of the population involved in the reinitialization (p_R) in CHC. The candidate values for the parameters were

- Population size, in GA and CHC: 60, 120, 200.
- Crossover probability, in GA and CHC: 0.5, 0.7, 0.9.
- Mutation probability, in GA: 0.01, 0.05, 0.1.
- Percentage of reinitialization, in CHC: 0.4, 0.6, 0.8.

In the parameters setting experiments, the best makespan results were obtained when using the following parameter configurations:

- **GA**: population size: 120, p_C: 0.7, p_M: 0.1.
- **CHC**: population size: 120, p_C: 0.7, p_R: 0.6.

Figures 4 and 5 present an example of the results obtained in a representative execution of the configuration setting experiments. The graphics report the average makespan values achieved for each combination of p_C and p_M (or p_R) in 20 executions of GA and CHC for solving the problem instance `u_i_hilo.0`, when using the stopping criterion of 90 s and a population size of 120 individuals.

The parameter configuration results demonstrate the importance of the mutation and reinitialization operators to achieve highly accurate solutions in short execution times: the best results are obtained when using (unusual) high mutation probability (in GA) and percentage of reinitialization (in CHC). In addition, no significant improvements were detected in the makespan values obtained by the sequential algorithms when increasing the population size from 120 to 200, suggesting that working with a larger population is not beneficial for the EAs: the processing time increases, thus conspiring against quickly achieving accurate results.

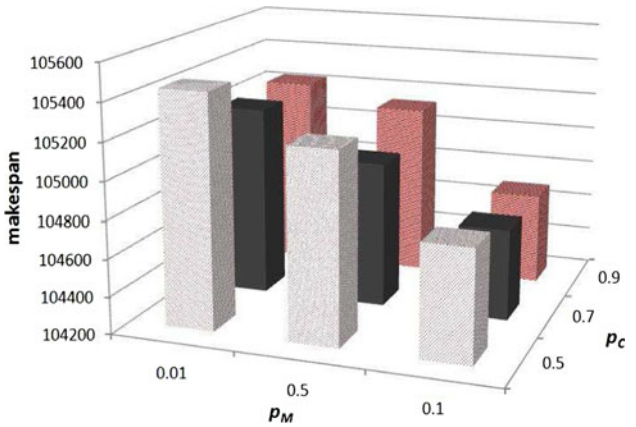

Fig. 4 GA parameter setting sample (`u_i_hilo.0`)

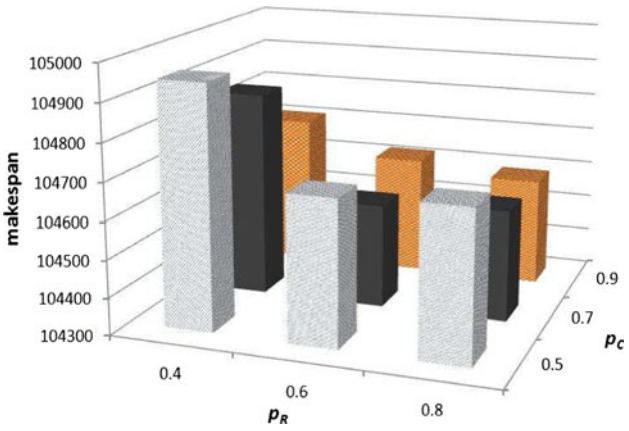

Fig. 5 CHC parameter setting sample (`u_i_hilo.0`)

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

Additional experiments were performed in order to find adequate values for the probabilities in the mutation and reinitialization operators. After performing the configuration analysis, the value of `HEAVY_MACH` (the probability of selecting the machine with the largest local makespan) was fixed at 0.7, and the value of `MAX_TRIALS` (number of attempts to find a better solution before accepting a worse one) was set at five.

5.4 Empirical analysis

This subsection presents the experimental results of applying the studied sequential and parallel EAs to solve the HCSP instances from Braun et al. Considerable effort has been made in this work to design accurate EAs for the HCSP, able to improve over the previously designed metaheuristics for this problem set. A comparison with both lower bounds and results obtained with other metaheuristics, a study of the fitness evolution and execution time, and a scalability analysis are also presented in this subsection.

5.4.1 Sequential algorithms

The results achieved using the sequential versions of GA and CHC are presented in Table 1. The table shows the best, average, and standard deviations (σ) on the makespan

Table 1 Results of sequential EAs for the HCSP

Instance	GA			CHC		
	Best	Avg.	σ (%)	Best	Avg.	σ (%)
u_c_hihi.0	7,659,878.7	7,699,080.1	0.41	**7,599,288.4**	7,681,050.1	0.55
u_c_hilo.0	155,092.0	155,300.1	0.11	**154,947.0**	155,333.4	0.19
u_c_lohi.0	**250,511.8**	252,568.7	0.56	251,194.3	251,868.3	0.22
u_c_lolo.0	5,239.1	5,248.6	0.16	**5,225.9**	5,241.9	0.20
u_i_hihi.0	3,019,844.3	3,030,564.2	0.22	**3,015,048.5**	3,024,904.9	0.30
u_i_hilo.0	**74,142.9**	74,568.4	0.41	74,240.9	74,375.9	0.12
u_i_lohi.0	104,688.0	105,048.1	0.31	**104,546.0**	104,939.1	0.32
u_i_lolo.0	2,577.0	2,587.8	0.46	**2,576.7**	2,582.2	0.13
u_s_hihi.0	4,332,248.2	4,347,835.5	0.38	**4,299,146.3**	4,320,803.4	0.52
u_s_hilo.0	**97,630.1**	98,026.1	0.41	97,888.2	98,307.4	0.27
u_s_lohi.0	126,438.0	126,840.8	0.25	**126,238.0**	126,580.4	0.20
u_s_lolo.0	3,510.4	3,516.5	0.23	**3,492.1**	3,505.0	0.25

Bold values are the best results obtained in each experiment

Table 2 Results of parallel EAs for the HCSP

Instance	Parallel GA			Parallel CHC		
	Best	Avg.	σ (%)	Best	Avg.	σ (%)
u_c_hihi.0	7,577,921.9	7,606,613.0	0.29	**7,461,819.1**	7,481,194.5	0.26
u_c_hilo.0	154,915.0	155,036.5	0.06	**153,791.9**	153,924.0	0.06
u_c_lohi.0	248,772.4	249,687.9	0.33	**241,513.2**	243,446.3	0.29
u_c_lolo.0	5,208.3	5,224.7	0.17	**5,177.5**	5,181.6	0.07
u_i_hihi.0	2,990,517.8	3,002,119.3	0.25	**2,952,493.2**	2,956,905.7	0.21
u_i_hilo.0	74,030.3	74,102.8	0.21	**73,639.8**	73,847.1	0.13
u_i_lohi.0	103,516.0	104,078.6	0.34	**102,123.1**	102,677.3	0.30
u_i_lolo.0	2,575.4	2,577.0	0.12	**2,548.9**	2,557.2	0.11
u_s_hihi.0	4,262,337.5	4,282,920.5	0.25	**4,198,779.5**	4,239,146.3	0.36
u_s_hilo.0	97,505.5	97,585.5	0.05	**96,623.3**	96,750.3	0.13
u_s_lohi.0	125,717.0	126,100.1	0.22	**123,236.9**	123,989.4	0.24
u_s_lolo.0	3,480.3	3,487.2	0.11	**3,450.1**	3,472.2	0.13

Bold values are the best results obtained in each experiment

values achieved in 30 independent executions of GA and CHC for solving each of the 12 problem instances studied.

The analysis of Table 1 indicates that CHC systematically achieved better results than GA for nine out of twelve problem instances studied (GA only obtained better makespan values for u_c_lohi.0, u_i_hilo.0, and u_s_hilo.0). These results suggest that the CHC evolutionary model (using HUX, diversity preservation, and population reinitialization), is a promising strategy for solving the HCSP when compared with a traditional GA model.

The results obtained using both sequential EAs outperformed previous results achieved with evolutionary techniques for inconsistent and semi-consistent problem instances. However, the sequential methods were not able to achieve the best makespan results formerly known for the problem instances from Braun et al., published in Ritchie and Levine (2004) and Xhafa et al. (2008b).

5.4.2 *Parallel algorithms*

The results achieved using the parallel versions of GA and CHC are presented in Table 2. The table shows the best, average, and standard deviations (σ) on the makespan values computed in 30 independent executions of GA and CHC for solving each of the twelve problem instances studied.

The comparison of Tables 1 and 2 shows that the parallel models of EAs are able to significantly improve over the

results of the sequential algorithms (achieving up to 4% of improvement factor for u_c_lohi.0). The parallel algorithms take advantage of the multiple search pattern and the increased diversity provided by the subpopulation model to improve the evolutionary search. In addition, since they work with a reduced population, the parallel EAs have a more focused processing capability, which allows them to achieve highly accurate results when using the predefined effort stopping criteria of 90 s of execution time. The standard deviation of the makespan values are very small, well below 0.5%, demonstrating a high robustness behavior when solving the HCSP. It can be expected that the parallel EAs will find accurate schedules in any single execution for HCSP scenarios that follow the ETC model by Ali et al. (2000).

Table 3 shows the percentage of improvement factors (*imp.*) obtained when using the parallel CHC over the other studied methods. The Kruskal–Wallis test was performed to analyze the time distributions reported in Tables 1 and 2, and the correspondent p values are presented for each problem instance and each pairwise algorithm comparison.

The parallel CHC makespan values significantly improve over the results achieved with the other studied EAs, and the computed p values are very small, so the improvements can be considered as statistically significant.

5.4.3 Comparison against the Min–Min heuristic

Table 4 compares the results obtained with the parallel CHC—the best method among the studied in this work—and the Min–Min heuristic, used to define the random initialization procedure in the proposed EAs.

Table 4 shows that parallel CHC obtained significant improvements with respect to the Min–Min makespan values (between 4.85 and 18.63%). The averaged makespan improvements were 8.51% for consistent instances, 12.14% for inconsistent instances, and 11.32% for semi-consistent instances. The overall makespan improvement was **10.66%** with respect to the Min–Min results. Parallel CHC achieved significant larger improvements for scenarios with high task heterogeneity (average improvement: 14.28%) than for scenarios with high low heterogeneity (average improvement: 7.04%), suggesting that the method is well suited for scheduling in highly heterogeneous environments.

Table 3 Improvement factors when using parallel CHC

Instance	GA		CHC		parallel GA	
	imp. (%)	p value	imp. (%)	p value	imp. (%)	p value
u_c_hihi.0	2.65	$<10^{-4}$	1.84	$<10^{-4}$	1.56	$<10^{-4}$
u_c_hilo.0	0.85	1×10^{-4}	0.75	6×10^{-3}	0.73	5×10^{-3}
u_c_lohi.0	3.73	$<10^{-4}$	4.01	$<10^{-4}$	3.01	$<10^{-4}$
u_c_lolo.0	1.19	$<10^{-4}$	0.93	1×10^{-4}	0.59	3×10^{-3}
u_i_hihi.0	2.28	$<10^{-4}$	2.12	$<10^{-4}$	1.29	1.10^{-4}
u_i_hilo.0	0.68	7×10^{-3}	0.82	1×10^{-4}	0.53	4×10^{-3}
u_i_lohi.0	2.51	$<10^{-4}$	2.37	$<10^{-4}$	1.36	$<10^{-4}$
u_i_lolo.0	1.10	$<10^{-4}$	1.09	$<10^{-4}$	1.04	$<10^{-4}$
u_s_hihi.0	3.18	$<10^{-4}$	2.39	$<10^{-4}$	1.51	$<10^{-4}$
u_s_hilo.0	1.04	$<10^{-4}$	1.31	$<10^{-4}$	0.91	2×10^{-4}
u_s_lohi.0	2.60	$<10^{-4}$	2.44	$<10^{-4}$	2.01	$<10^{-4}$
u_s_lolo.0	1.75	$<10^{-4}$	1.22	$<10^{-4}$	0.88	2×10^{-4}

Table 4 Comparison of parallel CHC and Min–Min results

Instance	Min–Min	Parallel CHC		imp.	
		Avg.	Best	Avg. (%)	Best (%)
u_c_hihi.0	8,460,674.0	7,481,195.0	7,461,819.1	11.58	**11.81**
u_c_hilo.0	161,805.4	153,924.8	153,791.9	4.87	**4.95**
u_c_lohi.0	275,837.3	245,163.3	241,513.2	11.12	**12.44**
u_c_lolo.0	5,441.4	5,181.6	5,177.5	4.77	**4.85**
u_i_hihi.0	3,513,919.3	2,956,906.0	2,952,493.2	15.85	**15.98**
u_i_hilo.0	80,755.7	73,847.1	73,639.8	8.55	**8.81**
u_i_lohi.0	120,517.7	102,123.1	102,123.1	15.26	**15.26**
u_i_lolo.0	2,785.6	2,559.0	2,548.9	8.14	**8.50**
u_s_hihi.0	5,160,343.0	4,259,146.1	4,198,779.5	17.46	**18.63**
u_s_hilo.0	104,375.2	96,750.3	96,623.3	7.31	**7.43**
u_s_lohi.0	140,284.5	124,760.7	123,236.9	11.07	**12.15**
u_s_lolo.0	3,806.8	3,476.4	3,450.1	8.68	**9.37**

Bold values are the best results obtained in each experiment

5.4.4 Comparison with lower bounds

The (non-preemptive) HSCP cannot be solved in reasonable execution times using exact methods, due to its high computational complexity. However, a lower bound for the makespan value can be computed by solving the linear relaxation for the preemptive problem version. Under the preemption hypothesis, the scheduler can temporarily interrupt a task, and continue its execution at a later time, without additional costs for the context switch procedure. In this (unrealistic) situation, an optimal solution has all machines with the same value of local makespan, which corresponds to the (optimum) global makespan of the schedule. This linear relaxation was solved using GLPSOL, the linear programming solver included in GLPK, the GNU Linear Programming Kit (Makhorin 2006), for the twelve problem instances studied.

The computed lower bounds are useful to determine the accuracy of the results achieved using the EAs proposed in this work.

Table 5 Comparison with the lower bounds for the preemptive case

Instance	Parallel CHC		LB	GAP	
	Avg.	Best		Avg. (%)	Best (%)
u_c_hihi.0	7,481,194.5	7,461,819.1	7,346,524.2	1.83	1.57
u_c_hilo.0	153,924.0	153,791.9	152,700.4	0.80	0.71
u_c_lohi.0	245,163.0	241,513.2	238,138.1	2.95	1.42
u_c_lolo.0	5,181.6	5,177.5	5,132.8	0.95	0.87
u_i_hihi.0	2,956,905.7	2,952,493.2	2,909,326.6	1.64	1.48
u_i_hilo.0	73,847.1	73,639.8	73,057.9	1.08	0.80
u_i_lohi.0	102,677.3	102,123.1	101,063.4	1.60	1.05
u_i_lolo.0	2,559.0	2,548.9	2,529.0	1.19	0.79
u_s_hihi.0	4,259,146.3	4,198,779.5	4,063,563.7	4.81	3.33
u_s_hilo.0	96,750.3	96,623.3	95,419.0	1.40	1.26
u_s_lohi.0	124,760.0	123,236.9	120,452.3	3.58	2.31
u_s_lolo.0	3,476.4	3,450.1	3,414.8	1.80	1.03

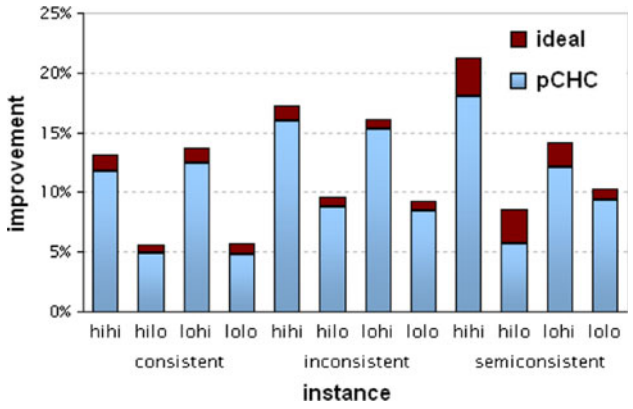

Fig. 6 Parallel CHC improvements with respect to Min–Min and LB

Table 5 shows a comparison between the average and best results achieved by parallel CHC—the best algorithm among the studied methods—and the lower bounds (LB) computed for the preemptive case. The table also reports the relative gap value of the best and average results achieved for each problem instance with respect to the correspondent lower bound (defined by Eq. 2).

$$\text{GAP} = \frac{\text{result} - \text{LB}}{\text{LB}} \tag{2}$$

The makespan values reported in Table 5 show that parallel CHC was able to achieve accurate results when compared with the (in the general case, unattainable) lower bounds for the preemptive case. The gaps were below 3.5% for all instances, and below 2% for ten of the twelve instances studied, suggesting that there is a small difference between the obtained results and the optimal makespan value for each problem instance.

The comparison against the results obtained with Min–Min and the computed LB for the preemptive case demonstrate that parallel CHC was able to achieve more than **80%** of the ideal improvement for the studied HCSP scenarios. Figure 6 summarizes the results, showing the improvements obtained by parallel CHC with respect to both the Min–Min results and the ideal improvements considering the computed lower bounds.

5.4.5 Comparison against other metaheuristics

Table 6 presents a comparison among the best results achieved for the set of instances from Braun et al., reporting the best results previously found with diverse metaheuristic techniques and those obtained with the parallel CHC, the best method among the studied in this work.

Table 6 Comparative results: metaheuristics for the HCSP

Instance	GA (Braun et al.)	GA (Xhafa et al.)	MA+TS (Xhafa)	cMA (Xhafa et al.)	ACO+TS (Ritchie, Levine)	TS (Xhafa et al.)	Parallel CHC (this work)
u_c_hihi.0	8,050,844.5	7,610,176.7	7,530,020.2	7,700,929.8	7,497,200.9	**7,448,640.5**	7,461,819.1
u_c_hilo.0	156,249.2	155,251.2	153,917.2	155,334.8	154,234.6	**153,263.3**	153,791.9
u_c_lohi.0	258,756.8	248,466.8	245,288.9	251,360.2	244,097.3	241,672.7	**241,513.2**
u_c_lolo.0	5,272.3	5,227.0	5,173.7	5,218.2	5,178.4	**5,155.0**	5,177.5
u_i_hihi.0	3,104,762.5	3,077,705.8	3,058,474.9	3,186,664.7	**2,947,754.1**	2,957,854.1	2,952,493.2
u_i_hilo.0	75,816.1	75,924.0	75,108.5	75,856.6	73,776.2	73,692.9	**73,639.8**
u_i_lohi.0	107,500.7	106,069.1	105,808.6	110,620.8	102,445.8	103,865.7	**102,123.1**
u_i_lolo.0	2,614.4	2,613.1	2,596.6	2,624.2	2,553.5	2,552.1	**2,548.9**
u_s_hihi.0	4,566,206.0	4,359,312.6	4,321,015.4	4,424,540.9	**4,162,547.9**	4,168,795.9	4,198,779.5
u_s_hilo.0	98,519.4	98,334.6	97,177.3	98,283.7	96,762.0	**96,180.9**	96,623.3
u_s_lohi.0	130,616.5	127,641.9	127,633.0	130,014.5	123,922.0	123,407.4	**123,236.9**
u_s_lolo.0	3,583.4	3,515.5	3,484.1	3,522.1	3,455.2	3,450.5	**3,450.1**

Bold values are the best results obtained in each experiment

The comparison with the ACO+TS from Ritchie and Levine (2004) and the TS from Xhafa et al. (2008b) is specially relevant, since those methods have obtained the previous best known makespan results to date for the HCSP instances studied.

The analysis of Table 6 shows that the parallel CHC method outperformed the ACO+TS by Ritchie and Levine (2004) in ten out of twelve HCSP instances and it also outperformed the TS by Xhafa et al. (2008b) in seven out of twelve HCSP instances. In addition, the parallel CHC was able to achieve schedules with better makespan values than the previously best-known solutions in six problem instances (the correspondent best makespan values are marked in bold in Table 6). The solutions (schedules) with the lowest makespan values obtained using parallel CHC for each problem instance are publicly reported in the HCSP website http://www.fing.edu.uy/inco/cecal/hpc/HCS.

Table 7 summarizes the information about whether the parallel CHC algorithm improves over the previous best-known methods, and it also states when the parallel CHC achieves a best known solution for each HCSP instance studied.

5.4.6 *Fitness evolution and execution time*

This subsection analyzes the fitness evolution and the trade-off between the solution quality obtained using parallel CHC and the required execution time. Table 8 presents the evolution of the makespan values when using 10, 30, and 60 s of execution time (the values correspond to average makespan results obtained in ten twenty executions of parallel CHC) for three representative scenarios with high task heterogeneity. The average improvements over the Min–Min results are also presented.

Table 8 shows that despite starting from worse solutions than the one computed by Min–Min, parallel CHC is able to find well-suited schedules in a short time.

For inconsistent instances, 10 s are enough to achieve 6% of makespan improvement, while for consistent and semi-consistent instances, 30 s are required to obtain more than 7% of improvement. When using 60 s of execution time, parallel CHC obtains an improvement of over 10% for all scenarios (and almost 15% for inconsistent instances). These results can be further improved by seeding the population with the full solution computed by the Min–Min heuristic.

Figures 7 and 8 show the evolution of the best makespan values observed for the CHC algorithms during representative executions over `u_i_hihi.0` and `u_s_lohi.0` (the makespan value of the Min–Min solution is included as a reference baseline). The slope of the curves show that the parallel model allows computing accurate schedules faster than the sequential CHC, which works slower when compared with the parallel CHC. The same behavior was verified for the parallel GA. These results confirm the

Table 7 Parallel CHC comparative results

Instance	Parallel CHC		
	Is better than		Achieves a best known solution
	ACO+TS (Ritchie, Levine)	TS (Xhafa et al.)	
`u_c_hihi.0`	**YES**	NO	NO
`u_c_hilo.0`	**YES**	NO	NO
`u_c_lohi.0`	**YES**	**YES**	**YES**
`u_c_lolo.0`	**YES**	NO	NO
`u_i_hihi.0`	NO	**YES**	NO
`u_i_hilo.0`	**YES**	**YES**	**YES**
`u_i_lohi.0`	**YES**	**YES**	**YES**
`u_i_lolo.0`	**YES**	**YES**	**YES**
`u_s_hihi.0`	NO	NO	NO
`u_s_hilo.0`	**YES**	NO	NO
`u_s_lohi.0`	**YES**	**YES**	**YES**
`u_s_lolo.0`	**YES**	**YES**	**YES**
TOTAL	**10/12**	**7/12**	**6/12**

Bold values are the best results obtained in each experiment

Table 8 Trade-off between solution quality and execution time

Instance	Min–Min	Parallel CHC		
		10 s	30 s	60 s
`u_c_lohi.0`	275,837.3	285,717.4	254,394.6	245,213.7
`u_i_hihi.0`	3,513,919.3	3,306,819.0	3,042,056.3	2,990,121.8
`u_s_lohi.0`	140,284.5	144,902.7	130,109.2	126,219.3

Instance	Min–Min	Improvement (%)		
		10 s	30 s	60 s
`u_c_lohi.0`		−3.6	7.77	11.10
`u_i_hihi.0`		5.9	13.43	14.91
`u_s_lohi.0`		−3.3	7.25	10.03

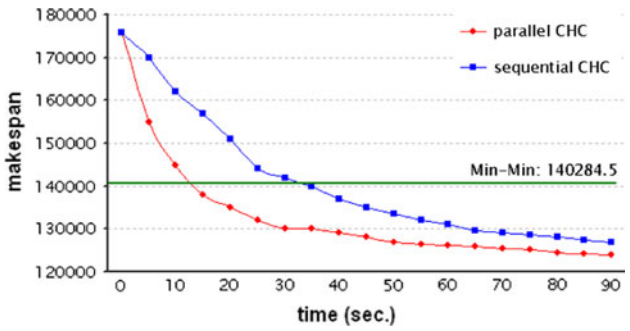

Fig. 7 Makespan evolution for CHC algorithms on `u_i_hihi.0`

ability of parallel EAs to obtain more accurate solutions than the sequential methods and also to reduce the required computing times.

5.4.7 Scalability analysis

This subsection presents a study of the parallel CHC scalability when solving large problem scenarios. In order to perform the study, three inconsistent HCSP instances with high task and machine heterogeneity were designed following the methodology proposed by Ali et al. (2000) for creating ETC matrices to model HC scenarios. These instances were selected since they represent the most general class of distributed HC systems, while they also have been characterized as "the most difficult instances to solve" by Xhafa et al. (2007). The HCSP instances considered in the scalability analysis have a dimension of (tasks × machines) 1,024 × 32, 2,048 × 64, and 4,096× 128.

Table 9 presents the best and average results obtained in 30 independent executions of the parallel CHC method for solving the large dimension inconsistent HCSP instances (the results for the `u_i_hihi.0` HCSP instance with dimension 512 × 16 are included in the table for comparative purposes). There have been no previous works solving the proposed large HCSP instances, so the parallel CHC results are compared with those achieved by the Min–Min deterministic heuristic. The improvements (in percentage) achieved by the best results obtained with parallel CHC over the Min–Min results are presented in the last column of the table for each problem instance.

The results presented in Table 9 show that the parallel CHC algorithm is able to obtain increasing improvements over the makespan results computed using the Min–Min deterministic heuristics as the size of the faced HCSP instances grow. When the problem complexity increases, the evolutionary search in parallel CHC computes accurate schedules that significantly outperform the Min–Min makespan results. Up to **23.27%** of improvement was achieved for the largest problem instance studied (dimension 4,096 × 128). Figure 9 summarizes the improvements obtained using the parallel CHC method.

The previous results show a good scalability behavior of the proposed parallel CHC algorithm when solving large inconsistent HCSP instances. Since these scenarios represent the most generic HC systems, this analysis suggests that parallel CHC is an efficient scheduler for large dimension scheduling problems arising in large clusters and medium-sized grid infrastructures.

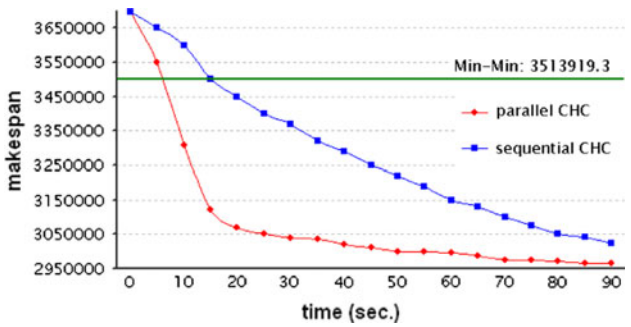

Fig. 8 Makespan evolution for CHC algorithms on `u_s_lohi.0`

Table 9 pCHC improvements over Min–Min

dimension	Min–Min	Parallel CHC		Improvement (%)
		Best	Average	
512 × 16	3,513,919.3	2,952,493.2	2,956,906.0	**15.98**
1,024 × 32	6,367,767.5	5,169,960.5	5,244,046.9	**18.81**
2,048 × 64	3,248,935.5	2,506,258.5	2,546,459.7	**22.86**
4,096 × 128	524,174.1	402,182.1	405,768.5	**23.27**

Bold values are the best results obtained in each experiment

6 Conclusions and future work

This work has presented advances on applying sequential and parallel EAs to the HCSP, a capital problem when executing tasks in heterogeneous distributed systems. The proposed EAs were designed for finding accurate results in an efficient way, using a predefined stopping criterion that allows a quick planning, and eventually rescheduling of incoming tasks. Fast parallel versions of GA and CHC were designed aimed at exploiting both the intrinsic parallel nature of EAs and the resource availability in distributed computing environments or in a modern multicore computer. The EAs were implemented using the MALLBA library, and they were executed in a high-performance cluster for solving a de-facto benchmark set with twelve HCSP instances. The analysis provides a first step to study the accuracy of

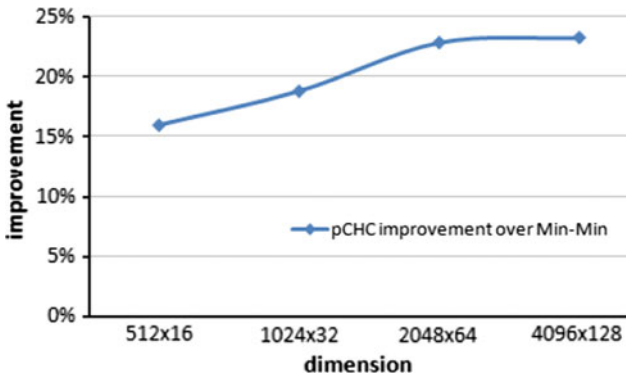

Fig. 9 pCHC improvements over Min–Min

sequential and parallel EAs to solve small HCSP instances, before scaling up to solve larger scenarios.

The analysis of the results allows drawing some conclusions on the applicability of EAs to solve the HCSP. The traditional GA is not well suited to efficiently solve the HCSP, mainly because its slow fitness evolution conspires against finding accurate results in short times. CHC is a more adequate strategy to solve the HCSP when compared with a traditional GA. The parallel EAs significantly improved the results of the sequential algorithms, by taking advantage of their multiple search mechanism and the diversity provided by the subpopulation model. The parallel version of CHC obtained accurate schedules in a low number of generations and improved its best fitness value faster than the other EAs. It achieved the best results among the studied EAs, obtaining an overall makespan improvement of **10.66%** with respect to the Min–Min results, and larger improvements (up to 18.63%) for scenarios with high task heterogeneity. The parallel CHC also improved the previously best-known solutions for six out of twelve HCSP instances studied. When comparing with lower bounds computed for the preemptive case, the gaps were below **3.5%** for all instances, suggesting that there is a small distance between the obtained results and the optimal makespan value for each instance. The parallel CHC was able to achieve more than **80%** of the ideal improvement for the studied scenarios.

The parallel algorithms were able to take advantage of working with small populations to improve the search efficacy and efficiency by using the available computing power in a parallel-distributed environment. The parallel CHC method showed good trade-off values when considering the solution quality and the execution time required to compute it. Despite starting from suboptimal schedules, it was able to find well-suited schedules in a reasonable short time (i.e. less than 10 s for inconsistent instances). A primary scalability analysis showed that the parallel CHC was able to compute accurate schedules for large inconsistent HCSP instances with high task and machine heterogeneity. Improvements up to **23.27%** over the Min–Min makespan results were obtained for problem instances with dimension (tasks $\times$ machines) $4{,}096 \times 128$.

From the previous results, we can claim that parallel CHC is a promising technique to use in large distributed HC and grid environments, when dealing with tasks having long execution times. In these scenarios, it is worth to invest the time required for computing the schedule (i.e. 1 min) in order to achieve significant improvements (**over 10%**) in the makespan values over traditional heuristics.

Two main lines remains to be tackled as future work, both already in progress: to improve the efficacy and efficiency of the proposed EAs, and to further study the ability of parallel EAs to solve large dimension HCSP instances. Improving the efficacy and efficiency implies studying additional operators able to provide the required diversity to avoid that the EAs get stuck when using a large number of small subpopulations, in order to reduce the execution times required to compute accurate schedules. In addition, the proposed methods should be applied to solve a more comprehensive set of large dimension HCSP instances, that model large HC and grid environments. By combining these two lines of future work, a special variant of the parallel CHC method using a powerful divergency operator could be enabled to significantly improve the search, allowing to face large dimension HCSP instances by using even one single multicore scheduling server.

Acknowledgments The work of S. Nesmachnow and H. Cancela has been partially supported by Programa de Desarrollo de las Ciencias Básicas, Comisión Sectorial de Investigación Científica, Universidad de la República, and Agencia Nacional de Investigación e Innovación, Uruguay. The work of E. Alba has been partially funded by the Spanish government and European FEDER through contract TIN2008-06491-C04-01 (M* project), and by the Andalusian government through contract P07-TIC-03044 (DIRICOM project).

References

Abraham A, Buyya R, Nath B (2000) Nature heuristics for scheduling jobs on computational grids. In: Proceedings of 8th IEEE international conference on advanced computing and communications, pp 45–52

Alba E (2005) Parallel metaheuristics: a new class of algorithms. Wiley, New York. ISBN 0471678066

Alba E, Tomassini M (2002) Parallelism and evolutionary algorithms. IEEE Trans Evol Comput 6(5):443–462

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

Ali S, Siegel H, Maheswaran M, Ali S, Hensgen D (2000) Task execution time modeling for heterogeneous computing systems. In: Proceedings of the 9th heterogeneous computing workshop, Washington, DC, USA, IEEE Computer Society, p 185

Bäck T, Fogel D, Michalewicz Z (eds) (1997) Handbook of evolutionary computation. Oxford University Press

Berman F, Fox G, Hey A (2003) Grid computing: making the global infrastructure a reality. Wiley, New York. ISBN 0470853190

Boyer W, Hura G (2005) Non-evolutionary algorithm for scheduling dependent tasks in distributed heterogeneous computing environments. J Parallel Distrib Comput 65(9):1035–1046

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

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

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

Davis E, Jaffe J (1981) Algorithms for scheduling tasks on unrelated processors. J ACM 28(4):721–736

Duran B, Xhafa F (2006) The effects of two replacement strategies on a genetic algorithm for scheduling jobs on computational grids. In: Proceedings of the 2006 ACM symposium on applied computing, New York. ACM, pp 960–961

El-Rewini H, Lewis T, Ali H (1994) Task scheduling in parallel and distributed systems. Prentice-Hall, Upper Saddle River. ISBN 0-13-099235-6

Eshaghian M (1996) Heterogeneous computing. Artech House, Norwood, MA, USA. ISBN 0-8906-552-7

Eshelman L (1991) The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination. In: Foundations of genetics algorithms. Morgan Kaufmann Publishers, Los Altos, pp 265–283

Foster I, Kesselman C (1998) The grid: blueprint for a future computing infrastructure. Morgan Kaufmann Publishers, Los Altos

Freund R, Sunderam V, Gottlieb A, Hwang K, Sahni S (1994) Special issue on heterogeneous processing. J Parallel Distrib Comput 21(3)

Garey M, Johnson D (1979) Computers and intractability. Freeman, New York

Goldberg D (1989) Genetic algorithms in search, optimization, and machine learning. Addison Wesley, New York

Grajcar M (1999) Genetic list scheduling algorithm for scheduling and allocation on a loosely coupled heterogeneous multiprocessor system. In: Proceedings of the 36th ACM/IEEE conference on design automation, New York. ACM, pp 280–285

Grajcar M (2001) Strengths and weaknesses of genetic list scheduling for heterogeneous systems. In: Proceedings of international conference on application of concurrency to system design, pp 123n++–132. IEEE

Ibarra O, Kim E (1977) Heuristic algorithms for scheduling independent tasks on nonidentical processors. J ACM 24(2):280–289

Kwok Y, Ahmad I (1997) Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. J Parallel Distrib Comput 47:58–77

Leung J, Kelly L, Anderson J (2004) Handbook of scheduling: algorithms, models, and performance analysis. CRC Press, Inc., Boca Raton. ISBN 1584883979

Li H, Groep D, Templon J, Wolters L (2004) Predicting job start times on clusters. In: Proceedings of the 2004 IEEE international symposium on cluster computing and the grid, Washington. IEEE Computer Society, pp 301–308

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

Makhorin A (2006) GNU linear programming kit, version 4.9. GNU Software Foundation. http://www.gnu.org/software/glpk/glpk.html

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

Wang L, Siegel H, Roychowdhury V, Maciejewski A (1997) Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. J Parallel Distrib Comput 47(1):8–22

Wu A, Yu H, Jin S, Lin K, Schiavone G (2004) An incremental genetic algorithm approach to multiprocessor scheduling. IEEE Trans Parallel Distrib Syst 15(9):824–834

Xhafa F (2007) A hybrid evolutionary heuristic for job scheduling in computational grids, chapter 10. Springer Verlag Series: studies in computational intelligence, vol 75

Xhafa F, Duran B (2008) Parallel memetic algorithms for independent job scheduling in computational grids. In: Cotta C, van Hemert J (eds) Recent advances in evolutionary computation for combinatorial optimization, volume 153 of Studies in computational intelligence. Springer, New York, pp 219–239

Xhafa F, Carretero J, Abraham A (2007) Genetic algorithm based schedulers for grid computing systems. Int J Innovative Comput Inf Control 3(5):1–19

Xhafa F, Alba E, Dorronsoro B, Duran B (2008a) Efficient batch job scheduling in grids using cellular memetic algorithms. J Math Model Algorithms 7(2):217–236

Xhafa F, Carretero J, Alba E, Dorronsoro B (2008b) Design and evaluation of tabu search method for job scheduling in distributed environments. In: Proceedings of the 21th international parallel and distributed processing symposium. IEEE Computer Society, pp 1–8

Xhafa F, Duran B, Abraham A, Dahal K (2008c) Tuning struggle strategy in genetic algorithms for scheduling in computational grids. In: Proceedings of the 7th Computer information systems and industrial management applications, Washington, DC, USA. IEEE Computer Society, pp 275–280

Yang J, Ahmad I, Ghafoor A (1993) Estimation of execution times on heterogeneous supercomputer architectures. In: Proceedings of the 1993 international conference on parallel processing, Washington, DC, USA, IEEE Computer Society, pp 219–226

Zomaya A, Teh Y (2001) Observations on using genetic algorithms for dynamic load-balancing. IEEE Trans Parallel Distrib Syst 12(9):899–911