Computational Intelligence
An International Journal

Computational Intelligence, Volume 28, Number 2, 2012

SCHEDULING IN HETEROGENEOUS COMPUTING AND GRID ENVIRONMENTS USING A PARALLEL CHC EVOLUTIONARY ALGORITHM

SERGIO NESMACHNOW,[1] ENRIQUE ALBA,[2] AND HÉCTOR CANCELA[3]

[1] *Universidad de la República, Herrera y Reissig 565, Montevideo, Uruguay*
[2] *Universidad de Málaga, Campus de Teatinos, Málaga, Spain*
[3] *Universidad de la República, Herrera y Reissig 565, Montevideo, Uruguay*

Scheduling is a capital problem when using distributed heterogeneous computing (HC) and grid environments to solve complex problems. The scheduling problem in heterogeneous environments is NP-hard, so a significant effort has been made to develop efficient methods for solving the problem. However, few works have faced realistic grid-sized problem instances. This work presents a parallel CHC (pCHC) evolutionary algorithm codified over MALLBA, a general-purpose library for combinatorial optimization, for solving the scheduling problem in HC and grid environments. Efficient numerical results are reported in the experimental analysis performed on both a standard benchmark and a set of large-sized problem instances specially designed in this work. The comparative study shows that pCHC is able to achieve high problem solving efficacy, significantly improving over traditional deterministic scheduling methods, while also showing a good scalability behavior when solving large problem instances.

Received 30 September 2009; Revised 23 June 2010; Accepted 24 June 2010; Published online 23 April 2012

Key words: grid, heterogeneous computing, parallel evolutionary algorithms, scheduling.

1. INTRODUCTION

Distributed computing environments have grown at a fast pace. Starting from small clusters of homogeneous computers in the 1980s decade, as for today they include platforms formed by hundreds or thousands of heterogeneous computing (HC) resources widespread around the globe. Nowadays, the expression *grid computing* denotes the set of distributed computing techniques that work over a large loosely coupled virtual supercomputer, formed by putting together many heterogeneous components of different characteristics and computing power. This infrastructure has made it feasible to provide pervasive and cost-effective access to distributed computing resources for solving hard problems (Foster and Kesselman 1998).

A crucial problem when using such HC environments consists in finding a *scheduling* strategy for a set of tasks to be executed on the system, to optimally assign the computing resources by satisfying some efficiency criteria. Scheduling problems on homogeneous multiprocessor systems have been widely studied in operations research, and numerous methods have been proposed for finding accurate schedules in reasonable times. However, in the 1990s decade the research community started to pay attention to scheduling problems on HC environments due to the popularization of distributed computing and the growing use of heterogeneous clusters. In the last ten years, a lot of effort has been made to study the scheduling problem on HC environments, because this platform provides the efficiency required for distributed and grid computing techniques.

Traditional scheduling problems are NP-hard (Garey and Johnson 1979). The research community has been searching for new scheduling techniques that are able to improve over the traditional exact ones, whose low efficiency often makes them useless in practice for

Address correspondence to S. Nesmachnow, Universidad de la República, Herrera y Reissig 565, Montevideo, Uruguay; e-mail: sergion@fing.edu.uy

solving large-dimension scheduling problems in reasonable times. In this context, ad hoc heuristic and metaheuristics techniques showed up as promising methods for solving the HC and grid scheduling problems. Although these methods do not guarantee success in computing an optimal solution for the problem, they get appropriate quasi-optimal schedules that satisfy the efficiency requirements for real-life scenarios, in reasonable times. Among a broad set of modern metaheuristic techniques for optimization, evolutionary algorithms (EAs) (Bäck, Fogel, and Michalewicz 1997) have emerged as flexible and robust methods for solving the *HC scheduling problem* (HCSP), achieving the high level of problem solving efficacy also shown in many other areas of application. To further improve the efficiency of EAs, parallel implementations became a popular option to speed up the search, allowing to reach high quality results in a reasonable execution time even for hard-to-solve optimization problems.

EAs and other metaheuristics have been applied to the HCSP in the last ten years. The proposals included Genetic Algorithms (GA) (Wang et al. 1997; Braun et al. 2001; Zomaya and Teh 2001; Xhafa et al. 2008b), Memetic Algorithms (MA) (Xhafa 2007), and cellular MA (cMA) (Xhafa, Alba, and Dorronsoro 2007a). 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) (Ritchie and Levine 2004) that took a long time—over 3.5 hours—to perform the search, and a hierarchic TS (Xhafa et al. 2008a) that used a predefined stopping criterion of 100 s. Despite the numerous proposals on applying EAs and other metaheuristics to the HCSP, there have been few works studying large-size and realistic instances in grid environments, mainly due to the inherent complexity of dealing with the underlying high-dimension optimization problem. The survey of related works showed that there do not exist standardized problem benchmarks (except a set of low-dimension, de facto standard problems by Braun et al. (2001)). Thus, there is still room to contribute in those lines of research, by studying highly efficient EA implementations, able to deal with large-size HCSP instances by using the computational power of parallel and distributed environments.

In a previous work (Nesmachnow 2009), the CHC evolutionary algorithm was identified as a promising method for solving low-dimension HCSP instances. In this line of work, the main contributions of this article are to introduce a new set of HCSP instances—designed by following a well-known methodology—far more complex than the existing ones in the present literature to model realistic HC environments such as large clusters and medium-size grid infrastructures, and then to study the scalability of a new efficient parallel CHC (pCHC) method for solving the new HCSP instances. The novelty of the proposed method is given by three main issues: the CHC evolutionary algorithm has been scarcely used in the past—far less than other traditional EAs, such as the classic genetic algorithm, specially when solving real-world complex applications like the HCSP; the parallel/distributed model of EAs that incorporates a different search pattern which allows increasing the diversity in the pCHC population and improving the results; and the physical distributed implementation of pCHC, which allows taking advantage of using parallel and distributed infrastructures to improve the results and the efficiency of the scheduling method.

The manuscript is structured as follows. Next section presents the HCSP formulation. Section 3 offers concepts about execution time estimation and introduces the new HCSP instances. Section 4 introduces the paradigm of evolutionary computation, presents the CHC algorithm, and describes the application of parallelism to EAs. Section 5 describes the implementation details of the pCHC, and also presents MALLBA, the public C++ algorithmic environment on which the algorithms was implemented. The discussion of the experimental analysis and results are presented in Section 6, while the conclusions and possible lines for future work are formulated in Section 7.

2. PROBLEM FORMULATION

Let an HC system composed by 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 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 a minimum time.

Using a computing facility does not come for free, so scheduling problems mainly concern about time. 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. 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 an *execution time function* $ET : P \times T \rightarrow \mathbf{R}^+$, where $ET(t_i, m_j)$ is the time required to execute the task t_i in the machine m_j,
- the goal of the HCSP is to find an assignment of tasks to machines (a function $f : T^N \rightarrow P^M$) that minimizes the *makespan*, defined in equation 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 previous 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 (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, and in *parameter sweep applications*, which are structured as a set of multiple experiments, each one executed with a different set of parameter values.

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. Static scheduling has its 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: the static results can be used as a reference baseline to determine if a dynamic scheduler is taking the right decisions about using the resources in the system. In addition, 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.

3. EXECUTION TIME ESTIMATION AND HCSP INSTANCES

Execution time estimation is a common technique applied to model the execution time of tasks on computers since the early 1990s (Yang, Ahmad, and Ghafoor 1993). It relies on estimation methods such as task profiling, benchmarking, and statistical analysis of

TABLE 1. Parameters of ETC Models.

	Task heterogeneity		Machine heterogeneity	
Model	low	high	low	high
Ali et al. (2000a)	$R_{task} = 10$	$R_{task} = 100{,}000$	$R_{mach} = 10$	$R_{mach} = 1{,}000$
Braun et al. (2001)	$R_{task} = 100$	$R_{task} = 3{,}000$	$R_{mach} = 10$	$R_{mach} = 1{,}000$

both submitted workloads and resource utilization, to provide an accurate prediction of the execution time for a given task on a specific machine. Researchers have stated that predicting the task execution times is useful to guide the scheduling in HC environments (Li et al. 2004). This section introduces the *expected time to compute* performance estimation model. It also discusses the HCSP instances already used in the related literature as well as the new set of instances specifically designed in this work. Finally, it briefly describes traditional scheduling heuristics using performance estimation, used as reference baseline to evaluate our proposal.

3.1. Expected Time to Compute Estimation Model

The *expected time to compute* (ETC) estimation model (Ali et al. 2000a) provides an estimation for the execution time of a collection of tasks in an HC system, taking into account three key properties: machine heterogeneity, task heterogeneity, and consistence. *Machine heterogeneity* evaluates the variation of execution times for a given task across the HC resources, while *task heterogeneity* represents the variation of the tasks execution times for a given machine. In addition, 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. Such a structured scenario captures the reality of many SPMD applications executing with local input data. An *inconsistent* ETC scenario lacks of structure among the computing demands of tasks and the computing power of machines, so a given machine m_j may be faster than m_k when executing some tasks, and slower for others. This category represents generic HC systems that receive many kinds of tasks. A *semi-consistent* ETC scenario models inconsistent systems that include a consistent subsystem (there is not a predefined structure on the whole sets of tasks and machines, but some of them behave like a consistent HC system).

Ali et al. (2000b) proposed two methods to design ETC matrices for representing diverse HCSP scenarios: the *range based* method and the *coefficient of variation* method. Both methods are equivalent, as they follow the same general procedure. The coefficient of variation method is flexible, because it allows using empirical probability distributions, but it also has a complex formulation. Instead, the range based method provides a simpler procedure for generating ETC scenarios. The range based method defines two ranges: $(1, R_{mach})$ and $(1, R_{task})$ for machine and task heterogeneity, respectively. Heterogeneity values for machines (τ_M) and tasks (τ_T) are randomly sampled using a uniform distribution, and the expected time to compute task i in machine j is calculated by $ET(i, j) = \tau_T(i) \times \tau_M(j)$. Ali et al. suggested using the parameter values presented in the first row of Table 1—selected to model relevant scenarios for the Management System for Heterogeneous Networks project (Hensgen et al. 1999)—to generate HCSP scenarios. However, no specific HCSP test suites were designed in that work.

3.2. HCSP Instances

The review of related work allowed to conclude that there do not exist standard benchmarks for the HCSP. A test suite of twelve randomly generated instances by Braun et al. (2001) has been used to evaluate heuristic and metaheuristic methods for the HCSP in many publications. Those instances have been also adopted as a reference for designing a complete set of large HCSP instances in this work, so they are commented in Section 3.2.1. Some other authors, like Page and Naughton (2005) generated their own test suites of random instances to evaluate heuristic algorithms, while arguing that "it is not clear what characteristics a typical task would exhibit. Several other works used randomly generated HCSP instances, but researchers did not often put much effort in describing the methodology employed for creating the scenarios. It is also worth noting that none of the previous proposals have scaled to the dimension of realistic grid environments, with the exception of Xhafa, Carretero, and Abraham (2007b), who generated four inconsistent ETC matrices with 4,096 tasks and 256 machines with high task and machine heterogeneity, arguing that "these are usually the most difficult instances to solve. However, Xhafa et al. did not publish the HCSP instances used.

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

3.2.1. HCSP Instances from Braun et al. (2001). Braun et al. (2001) presented an HCSP test suite with twelve instances generated using the range based method. All the instances have 512 tasks and 16 machines, and they combine the three ETC model properties (task and machine heterogeneity, and consistency). Instead of using the previously proposed ETC parametrization by Ali et al. (2000a), the authors suggested the upper bounds for machine and task heterogeneity intervals presented in the second row of Table 1. These values were selected to model several characteristics of the prediction methods used in Armstrong, Hensgen, and Kidd (1998), but the authors pointed out that their election was quite arbitrary, and suggested that researchers may use their own values to generate instances that model other specific situations of interest. However, the commented test suite has become a de facto standard benchmark to evaluate algorithms for solving the HCSP.

The name of HCSP instances from Braun et al. (2001) 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).

3.2.2. Grid Infrastructures. The HCSP instances from Braun et al. (2001) were conceived for modeling multiprocessor HC systems, and so they do not capture the reality of nowadays grid infrastructures. The Berkeley Network of Workstations project surpassed the one-hundred-processors milestone in the middle of the 1990s decade (Anderson et al. 1995). As for today, modern grid initiatives use platforms with more than 1,000 processors, while hierarchical worldwide computing grids and volunteer-based distributed computing platforms manage more than 100,000 computing resources (see Table 2 for a brief description of sampled grid infrastructures).

EELA-2 (eScience GRID facility for Europe and Latin America) and Grid500 are examples of medium-sized grids. EELA-2 is the largest grid initiative involving Latin America

TABLE 2. Details of Sampled Grid Computing Infrastructures.

Size/type	Grid	Location	Processors	Comment/URL
medium-sized grids	EELA	Europe-LA	∼750	regional grid, `www.eu-eela.eu`
	Grid5000	France	>3,000	national grid, `www.grid5000.fr`
large grids	OSG	USA	>30,000	national grid, `www.opensciencegrid.org`
	TeraGrid	USA	>40,000	national grid, `www.teragrid.org`
	EGEE	Europe	>80,000	continental grid, `www.eu-egee.org`
	WLCG	Europe	>100,000	CERN worldwide grid, `www.cern.ch/lcg`

(Brasilero et al. 2008). Grid500 has 1,597 nodes (8 families, 17 systems) with a total number of 3,000 processors, but the experimental scenarios for scheduling algorithms usually consider less than 250 processors, often grouped in few heterogeneous classes (Caniou and Gay 2008; Mohamed and Epema 2008). On the other hand, very large grid infrastructures, such as TeraGrid and WLCG—the largest computing grid in the world in 2009, with almost 100,000 projected processors at CERN, Switzerland, and more than 100,000 additional processors distributed worldwide—have a hierarchical structure. In this large-scale organizations, heterogeneity is only handled on high-level schedulers, while local schedulers perform the tasks allocation intra-site, dealing with homogeneous machines.

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

3.2.3. New HCSP Instances. Apart from the proposals previously commented in Section 3.2.1, there has been little effort to define a standard test suite for HC scheduling. Even today, when grid scheduling has been the focus of many works, researchers have been using the test suite from Braun et al. (2001) or proprietary instances, often generated without following a methodological basis. One of the main objectives of this work consists in studying the scalability of new proposed methods to solve the HCSP (i.e., how the solution quality achieved using a fixed execution time varies when the instances dimension grows). To perform the analysis, this work introduces a test suite of large HCSP instances designed following the methodology for execution time estimation proposed by Ali et al. (2000a). The scenarios were created using a random generator program, implemented in the C language using the standard C libraries `stdlib.h` and `math.h`, without requiring any additional software. The generator implements the range based method from Ali et al. (2000a), regarding the relevant scenario parameters: dimension (number of tasks and machines), task and machine heterogeneity, consistency, and two parametrization models of ETC. The output file format is identical to the one employed by Braun et al. (2001): a column vector of $N \times M$ floating point numbers that represents the ETC matrix, ordered by task identifier.

The test suite generated in this work includes HCSP instances with diverse complexity. The *small-sized* instances extend Braun et al.'s problems up to 1,024 tasks and 32 processors. The *medium-sized* instances include up to 4,096 tasks and 128 machines, and are considered as representative of large multiprocessor, medium-size clusters of computers, and small grid systems. The group of *large-sized* instances include scenarios with up to 8,192 tasks and 256 processors, a dimension that represents large clusters and medium-size grid systems. For each dimension, twenty-four HCSP instances were generated regarding all the heterogeneity and consistency combinations, twelve of them using the parameter values from Ali et al.,

and twelve using the values from Braun et al. (trying to avoid biased results). The instances are named following the previously commented convention: the names have the pattern M.d_c_MHTH, where the first letter (M) describes the heterogeneity model used (A for Ali, and B for Braun). The number 0 in the last position of the name was removed.

Because the new test suite was designed following a well-known methodology, the problem instances maintain the relevant properties of the ETC model by Ali et al. (2000a). By including a more comprehensive set of scenarios, it allows performing studies aimed at obtaining a better characterization of new scheduling methods. In addition, the new set of large-sized HCSP instances poses a real challenge to scheduling methods. The size of the search space increases from 3.2×10^{616} possible schedules in the set of instances by Braun et al. to 2×10^{19728} for the largest new HCSP instances, thus it is useful to analyze the efficacy of scheduling methods when the problem instances grow.

The problem instances and the generator program are publicly available to download at the HCSP website http://www.fing.edu.uy/inco/grupos/hpc/HCSP.

3.3. Scheduling Heuristics Using Performance Estimation

The class of *list scheduling techniques* (Schutten 1996; Kwok and Ahmad 1999) comprises a large set of deterministic static scheduling methods that work by assigning priorities to tasks based on a particular ad hoc heuristic. After that, the list of tasks is sorted in decreasing priority and each task is assigned to a processor, regarding the task priority and the processor availability. Algorithm 1 presents the general schema of a list scheduling method.

Algorithm 1 Schema of a list scheduling algorithm.

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

Since the pioneering work by Ibarra and Kim (1977), where the first algorithms following the generic schema presented in Algorithm 1 were introduced, many list scheduling techniques have been proposed to provide easy methods for tasks-to-processors scheduling. Three of them have been used in this work to provide a baseline for comparing the results achieved when using the proposed parallel evolutionary scheduling method:

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 ETC for that task.

Sufferage identifies the task that if it is not assigned to a certain host, will *suffer* the most. The *sufferage value* is computed as the difference between the best MCT of the task and its second-best MCT, and the 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 best machine. The mapped task is removed from U, and the process is repeated until all tasks are mapped. Min-Min improves upon the MCT heuristic, because 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 generally allows finding smaller makespan values than other heuristics, because more tasks are expected to be assigned to the machines that can complete them the earliest.

These heuristics have also been used to design probabilistic methods for the initialization procedure in our pCHC method (see Section 5.2).

4. EVOLUTIONARY ALGORITHMS

EAs are stochastic methods that emulate the evolutionary process of natural species to solve optimization, search, and learning problems (Goldberg 1989; Davis 1991). EAs have been successfully applied for solving optimization problems underlying many real applications of high complexity.

EAs are iterative techniques (each iteration is called a *generation*) that apply stochastic operators on a pool of individuals (the population P) 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 method. An evaluation function associates a fitness value to every individual, indicating its suitability to solve the problem. Iteratively, the application of *reproduction operators* like the *recombination* of parts of two individuals or random changes in their contents (*mutations*) is guided by a selection-of-the-best technique to better tentative solutions. 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). Finally, the best solution ever found in the iterative process is returned, taking into account the fitness function considered.

4.1. The CHC Evolutionary Algorithm

The CHC acronym stands for "Cross generational elitist selection, Heterogeneous recombination, and Cataclysmic mutation" (Eshelman 1991). CHC uses an elitist selection strategy that tends to perpetuate the best individuals in the population, and a special mating that only allows to reproduce those individuals which differ from each other by some number of bits. The initial threshold for allowing mating is often set to one-fourth 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. CHC does not apply mutation; diversity is provided by applying a re-initialization procedure, using the best individual found so far as a template for creating a new population after convergence is detected.

Algorithm 2 presents a pseudo-code for the CHC algorithm, based on Eshelman's proposal, showing those features that make it different from traditional EAs: the highly elitist

replacement strategy, the use of its own HUX 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.

Algorithm 2 Schema of the CHC algorithm.

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

4.2. 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 computing 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 pCHC proposed in this work is categorized within the *distributed subpopulations* model, according to the classification from Alba and Tomassini (2002): the original population is divided in several subpopulations (*demes*), separated geographically from each other. Each deme runs a serial CHC, 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 2 shows the generic schema for the migration procedure employed in a distributed subpopulation PEAs. Two conditions control the migration procedure: *sendmigrants* determines when the exchange of individuals takes place, and *receivemigrants* 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 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 3 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
```

5. A PARALLEL CHC FOR THE HCSP

The pCHC proposed in this work was designed to achieve accurate HCSP solutions in reduced time, while providing a good exploration pattern that allows solving large-sized instances. The implementation details are presented in this section, along with the software library in which the EAs were implemented.

5.1. The MALLBA Library

MALLBA (Alba et al. 2002) is a library of optimization algorithms that can deal with parallelism in a user-friendly and, at the same time, efficient manner. The pCHC described in this section is implemented as a generic template on the library as *software skeletons*, to be instantiated with the features of the problem by the user. Each skeleton incorporates all the knowledge related to the resolution method, its interactions with the problem, and the parallel considerations. In MALLBA, 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` (setup 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`. Using MALLBA allows a quick coding of the pCHC method to cope with the inherent difficulties of the HCSP.

5.2. Implementation Details

This section describes the problem encoding, the initialization, and the variation operators used in pCHC.

5.2.1. Problem Encoding. Two main alternatives have been proposed in the related works for encoding HCSP solutions: the *task oriented* and the *machine oriented* encoding. The first encoding uses a vector of machine identifiers to represent the task-to-resource

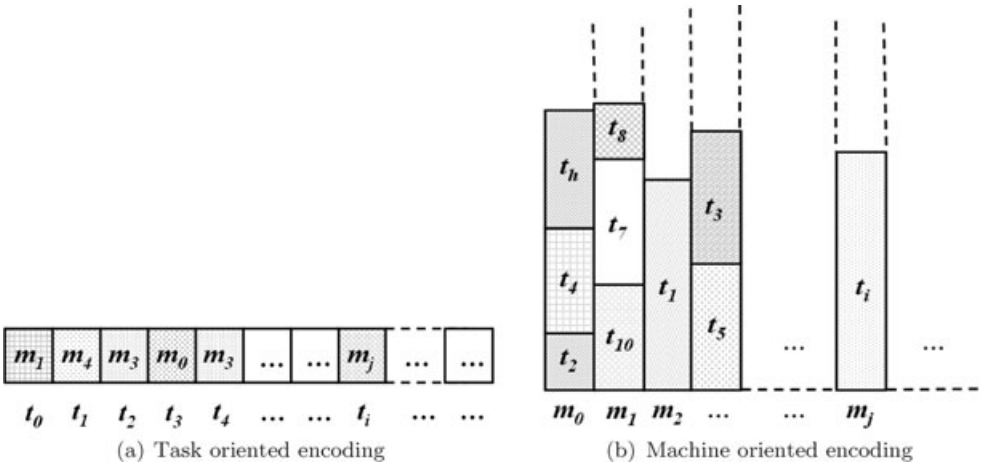


FIGURE 1. HCSP encodings.

assignment (see Figure 1 (a)). The presence of m_j in the position t_i means that the task t_i is scheduled to execute on machine m_j. This is a direct representation for schedules, which allows a straightforward exploration by using simple move-and-swap operators. However, it does not provide an easy way to evaluate the makespan variation after applying a move or swap operator, so any single change on a task assignment forces to perform a (highly inefficient) reevaluation of the whole schedule. The machine oriented encoding uses a 2D structure to represent the group of tasks scheduled to execute on each machine (see Figure 1(b)). It provides an easy and efficient way for performing move-and-swap exploration operators, because it is able to store specific metric values for each machine (such as the local makespan). Therefore, any single change on a task assignment does not imply reevaluating the schedule metric.

In a previous CHC implementation (Nesmachnow 2009), the task-oriented encoding was used to provide a simple method to solve the HCSP. In this work, the pCHC adopted the machine-oriented encoding, because storing the local makespan values significantly enhances the search efficiency.

5.2.2. Initialization. Numerous methods have been proposed to generate the initial population when applying EAs to the HCSP (Braun et al. 2001; Xhafa et al. 2007a,b; Xhafa and Duran 2008). Many of them employed specific heuristics to start the search from a set of useful suboptimal schedules, increasing the EA effectiveness to minimize the makespan.

In this work, several methods were studied to generate accurate initial solutions to speed up the search. When dealing with low-dimension HCSP instances, deterministic heuristics such as Min-Min and Sufferage provide accurate and easy-to-compute solutions to seed the population. Min-Min has been identified as an efficient method for finding accurate schedules for small size HCSP instances (Braun et al. 2001), and also when the ETC matrix has reasonable variations on heterogeneity (Luo, Lü, and Shi 2007), while Sufferage often achieved better schedules than Min-Min for inconsistent scenarios. However, when the problem dimension grows, the time required to compute the initial solution increases, thus reducing the EA efficiency. To avoid the performance degradation, probabilistic versions of Min-Min and Sufferage heuristics are used in this work for the population initialization: they follow the general procedure of the deterministic heuristic, but only for assigning a random number of MAX_{init} tasks, while the remaining tasks are assigned using a MCT strategy.

5.2.3. Recombination. The pCHC uses HUX to recombine the features of two solutions. The HUX implementation is straightforward: for each task, the corresponding machine in each offspring is chosen with uniform probability (0.5) between the two machines for that task within the parents' encoding. This operator is performed in linear order with respect to the number of tasks in the encoding (O(N)).

5.2.4. Reinitialization. The reinitialization performs small perturbations in a given schedule, aimed at providing diversity to the population, to avoid the search from getting stuck in local optima. It performs simple moves and swaps of tasks between two machines, selecting with high probability the machines with the highest and the 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 convergence is detected.

The reinitialization operator cyclically performs a maximum number of `MAX_TRIALS` 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 presented in related works by Xhafa et al. (2007b); Xhafa et al. (2008a)), 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 pCHC 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 pCHC to show a good scalability behavior when solving large HCSP instances.

6. EXPERIMENTAL EVALUATION

This section presents the experimental evaluation of pCHC, aimed at studying the efficiency of the proposed EA for solving the HCSP instances described in Section 3.2, and also analyzing the scalability behavior when solving large-sized problem instances. The section introduces the computational platform used for the experimental evaluation. After that, it presents and discusses the pCHC parameter settings experiments and the results obtained when solving the set of problem instances described in Section 3.2, including the numerical results, a comparison with other techniques, a scalability analysis, and a study of the parallel performance of pCHC.

6.1. Execution Platform

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

6.2. Parameter Settings

The main objective of this work consists in studying the ability of EAs to *efficiently* solve the HCSP, thus demonstrating their usefulness to act as a practical scheduler for real-life HC and grid systems. Therefore, the pCHC method uses a bounded time effort stopping criterion. This decision is useful for efficiently solving static HCSP instances, and is also effective for solving dynamic scenarios following the rescheduling strategy by replanning the incoming (and also unexecuted) tasks after certain intervals of time. When dealing with small and medium-sized HCSP instances, the stopping criterion was fixed at 90 s. of wall-clock time (following related works by Xhafa et al.), while when solving the large HCSP instances the stopping criterion was set at 120 s. of execution time.

The global population size was fixed at 120 individuals, and the pCHC worked with 8 subpopulations (15 individuals per deme), following the parameter settings experiments from a previous work (Nesmachnow 2009). A configuration analysis was performed to determine the optimum parameter values for each operator, using a subset of six low-dimension HCSP instances with diverse characteristics. In the experiments, the crossover probability (p_C) and the percentage of the population involved in the reinitialization (p_R) were selected considering the candidate values p_C: 0.5, 0.7, 0.9, and p_R: 0.4, 0.6, 0.8.

The best makespan results were obtained when using the parameter values p_C=0.7 and p_R=0.8, showing the importance of the reinitialization operators to achieve highly accurate solutions in short execution times (the best results were obtained when using an unusually high percentage of reinitialization). Additional experiments were performed to tune the reinitialization operator: the probability of selecting the machines with highest and lowest local makespan was set at 0.7, and the value of `MAX_TRIALS` was set at five.

6.3. Results and Discussion

This section presents the experimental results obtained with pCHC for solving the HCSP. The results for the set of instances from Braun et al. are reported separately, as there have been antecedents of solving the benchmark using other methods. After that, the pCHC results when solving the large-dimension problem instances—specially designed in this work—are presented and discussed. Finally, the section includes a study of the parallel performance and scalability of pCHC when solving large-sized HCSP instances.

6.3.1. Results for the Instances from Braun et al. (2001). Table 3 reports the results obtained in 30 independent executions of pCHC for the set of instances by Braun et al., and presents a comparison with previous techniques. In those cases where pCHC outperformed previous best results achieved with other techniques, Table 3 also presents the time required to achieve the previous best-known makespan value (t_B, in seconds), and the quantile of the empirical distribution of pCHC that outperformed the previous best method (i.e., the percentage of executions that obtained better makespan values than the previous best result reported, q).

The analysis of Table 3 shows that pCHC outperformed the results obtained with previous EAs. pCHC also outperformed the ACO+TS by Ritchie and Levine (2004) in ten out of twelve HCSP instances, and the TS by Xhafa et al. (2008a) in seven out of twelve HCSP instances. In addition, pCHC was able to achieve better makespan values than the previously best-known solutions in six problem instances (the correspondent makespan values are marked with bold in Table 3). Short execution times are required to outperform the previous results in those cases. The solutions (schedules) with the lowest makespan values obtained for each problem

TABLE 3. Comparative Results: Metaheuristics for the HCSP.

Instance	GA (Braun et al.)	MA+TS (Xhafa)	cMA (Xhafa et al.)	ACO+TS (Ritchie, Levine)	TS (Xhafa et al.)	pCHC best	pCHC avg.	pCHC σ	pCHC t_B	pCHC q
u_c_hihi.0	8050844.5	7530020.2	7700929.8	7497200.9	7448640.5	7461819.1	7481194.5	0.26%	–	–
u_c_hilo.0	156249.2	153917.2	155334.8	154234.6	153263.3	153791.9	153924	0.06%	–	–
u_c_lohi.0	258756.8	245288.9	251360.2	244097.3	241672.7	**241524.0**	243446.3	0.29%	71	0.33
u_c_lolo.0	5272.3	5173.7	5218.2	5178.4	5155.0	5177.5	5181.6	0.07%	–	–
u_i_hihi.0	3104762.5	3058474.9	3186664.7	2947754.1	2957854.1	2952493.2	2956905.7	0.21%	–	–
u_i_hilo.0	75816.1	75108.5	75856.6	73776.2	73692.9	**73639.8**	73847.1	0.13%	74	0.20
u_i_lohi.0	107500.7	105808.6	110620.8	102445.8	103865.7	**102136.1**	102677.3	0.30%	31	0.73
u_i_lolo.0	2614.4	2596.6	2624.2	2553.5	2552.1	**2549.8**	2557.2	0.11%	62	0.36
u_s_hihi.0	4566206	4321015.4	4424540.9	4162547.9	4168795.9	4198779.5	4239146.3	0.36%	–	–
u_s_hilo.0	98519.4	97177.3	98283.7	96762	96180.9	96623.3	96750.3	0.13%	–	–
u_s_lohi.0	130616.5	127633	130014.5	123922	123407.4	**123251.5**	123989.4	0.24%	55	0.16
u_s_lolo.0	3583.4	3484.1	3522.1	3455.2	3450.5	**3450.1**	3472.2	0.13%	80	0.13

TABLE 4. pCHC Results for New HCSP Instances with Dimension 1024×32.

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

instance are reported in the Appendix and further details are provided at the HCSP website (`http://www.fing.edu.uy/inco/cecal/hpc/HCSP`).

6.3.2. Results for the New Large-Sized HCSP Instances. Tables 4 to 7 show the pCHC best results for large HCSP instances. For each dimension, the results for the twelve instances following the heterogeneity model from Ali et al., and the twelve ones following the heterogeneity model from Braun et al. are presented.

There have been no previous works solving this new set of HCSP instances, so the pCHC results are compared with those achieved by the most popular static heuristics (MCT, Min-Min, and Sufferage). The tables report the makespan values obtained with the three deterministic heuristic, the best, average and standard deviation results achieved using pCHC in 30 independent executions, and the improvement factor (impr., in percentage) over the best deterministic heuristic result. The Kolmogorov–Smirnov and the Shapiro–Wilk test were applied to analyze the distribution of the pCHC results, and the observed significance levels for both tests allow to conclude that the empirical distribution of pCHC results follows a normal distribution with a confidence level of 95% for all the problem instances studied.

The pCHC achieved significant makespan improvements—ranging from 4.92% to 22.12%—over the best result provided by the list scheduling heuristics. It took advantage of

TABLE 5. pCHC Results for New HCSP Instances with Dimension 2048×64.

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

the multiple search pattern and the increased diversity provided by the subpopulation model to achieve high quality results. The standard deviation of the makespan values were very small (below 1.5%), demonstrating a high robustness behavior when solving the HCSP. It can be expected that pCHC will find accurate schedules in any single execution for ETC-based HCSP scenarios.

Regarding inconsistent and semiconsistent instances, pCHC obtained a roughly 10% makespan improvement factor over the best traditional heuristic, even for the larger instances. The classic methods provide accurate schedules for inconsistent HCSP instances with low dimension, so the pCHC improvements were slightly below 10% for those scenarios. However, pCHC notably improves over the traditional methods for larger inconsistent HCSP instances, achieving makespan reductions above 10% for all problem instances, and a maximum of 22.12% for A.u_i_hihi with dimension 2048×64. The opposite situation happened for consistent instances, where pCHC improvements diminished from nearly 10% to 5% as the instances dimension grows.

Table 8 summarizes the (percentage) averaged improvements when using pCHC with respect to the best traditional heuristic for each type of problem instances and dimensions studied, and Figure 2 shows the graphical comparison of average improvement values. The

TABLE 6. pCHC Results for New HCSP Instances with Dimension 4096×128.

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

results show that consistent instances are the most difficult to solve with pCHC, while it achieved accurate values for inconsistent and semiconsistent problem instances. The improvement values tend to slightly decrease as long as the problem instances grow, but they always remain above 5% for consistent instances, above 8% for inconsistent, and above 10% for semiconsistent instances.

The list scheduling heuristics compute their solutions in a (deterministic) time, which ranged from 10 s. (for dimension 2048×64) to 70 s. (for dimension 8192×256). From the previous results, we can claim that pCHC is a promising technique 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 these scenarios, it is worth to invest the time required for computing the schedule (i.e., two minutes) to achieve significant improvements (over 10%) in the makespan values over traditional heuristics.

6.3.3. Fitness evolution and execution time. This subsection analyzes the fitness evolution of the pCHC algorithm for two representative HCSP instances. The analysis allows drawing some conclusions about the trade-off between the solution quality obtained using pCHC and the required execution time.

TABLE 7. pCHC Results for New HCSP Instances with Dimension 8192×256.

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

TABLE 8. Improvement of pCHC over Traditional Heuristics.

		Dimension				
Model	Type	512×16	1024×32	2048×64	4096×128	8192×256
Ali et al.	consistent	11.34%	9.05%	6.98%	5.93%	5.04%
	inconsistent	9.85%	8.19%	15.22%	13.86%	11.46%
	semiconsistent	14.31%	12.28%	12.03%	11.14%	9.28%
Braun et al.	consistent	8.51%	8.24%	6.21%	6.11%	4.90%
	inconsistent	7.53%	9.26%	13.35%	13.08%	12.93%
	semiconsistent	11.32%	12.43%	11.95%	10.68%	9.64%

Figures 3 (a) and 3(b) show the evolution of the best makespan values observed for the pCHC algorithm during representative executions over `u_i_hihi.0` (dimension 2048×64) and `u_s_lohi.0` (dimension 4096×128) (the makespan value of the Min-Min and Sufferage solutions are included as a reference baseline). The slopes of the curves in Figure 3 show that despite starting from worse solutions than the one computed by Min-Min and Sufferage,

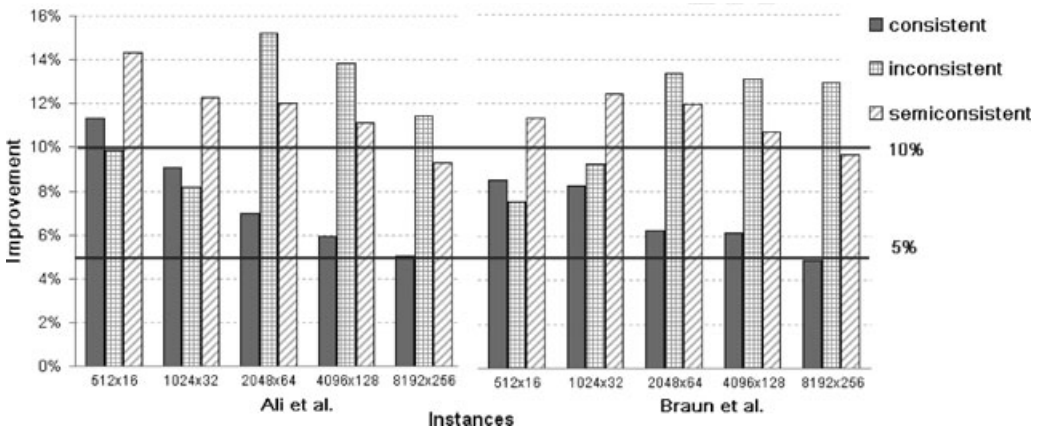
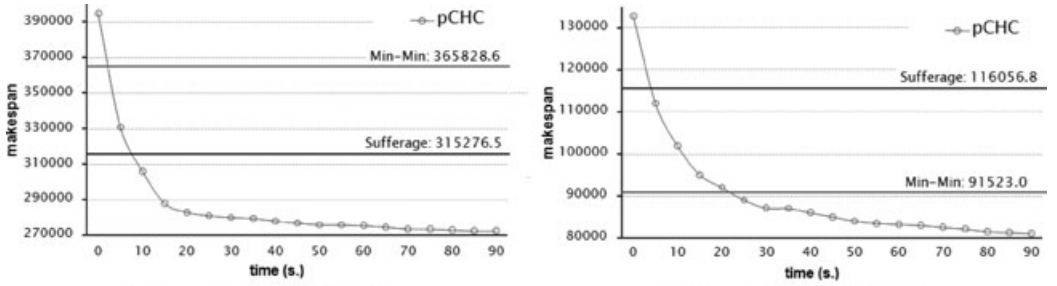


FIGURE 2. Improvement over traditional heuristics (percentage).



(a) A.u_i_hilo (2048×64). (b) B.u_s_lohi (4096×128).

FIGURE 3. Makespan evolution sample for pCHC.

pCHC is able to find well-suited schedules in a short time. Similar results were obtained for the other problem instances studied. For the small HCSP instances, pCHC improves over the traditional heuristic results in a few seconds, while almost 60 s. are required to obtain 5% of improvement in the largest instances.

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

6.3.4. Scalability Analysis. Figure 5 presents the results obtained in a scalability analysis devoted to study the behavior of pCHC when solving HCSP instances of increasing size. Mean values of the average *normalized* makespan (i.e., the quotient between the makespan achieved by pCHC using eight subpopulations and the makespan obtained with a panmictic population) were evaluated for consistent, inconsistent and semiconsistent instances for each problem dimension studied. The graphic shows that the normalized makespan values diminishes when solving large-dimension problem instances, specially for inconsistent and semiconsistent instances. Reductions of up to 10% in the mean makespan values were achieved when solving the largest problem instances (8192×256, semiconsistent instances), while the baseline results were around 3%–5% for the smaller problem instances (512×16). These results show that the parallel model allows pCHC to show a good scalability behavior,

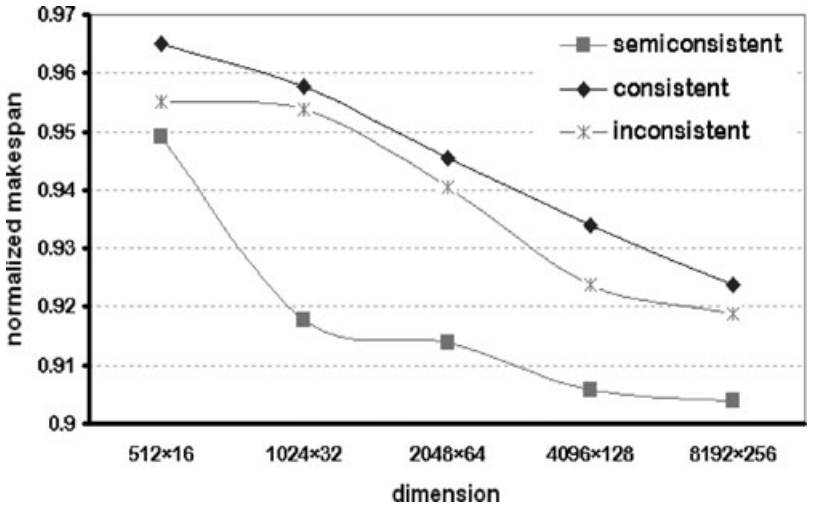


FIGURE 4. Scalability analysis for pCHC.

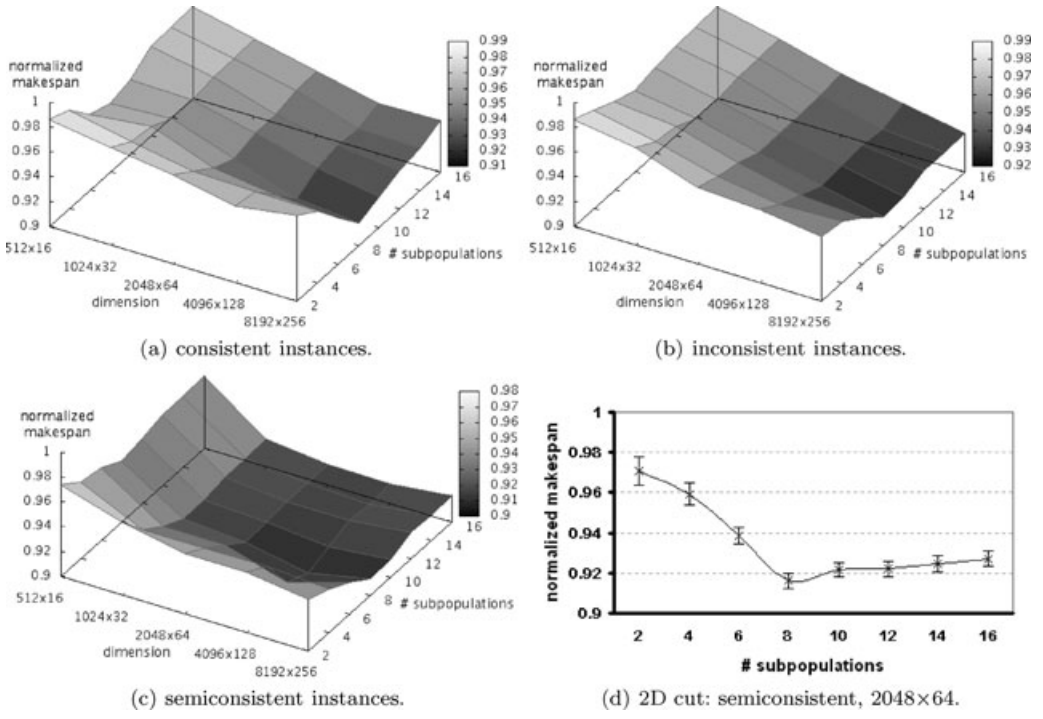


FIGURE 5. Parallel performance and scalability of pCHC.

achieving improved makespan reductions with respect to a panmictic model as the search space dimension grows.

6.3.5. Parallel Performance. All the previously reported results were obtained using eight subpopulations in pCHC, according to the parameter setting experiments performed on

small-sized HCSP instances. However, the parallel model should achieve improved results by splitting the population in more demes when facing the large HCSP instances. This subsection presents a study that evaluates the parallel performance of pCHC for solving HCSP instances with increasing dimension, eventually using additional computing resources available.

Instead of working with a fixed number of demes, the global population of 120 individuals was split into 2 to 16 subpopulations, and pCHC was executed until reaching the time stopping criterion. Mean values of the average *normalized* makespan improvements were evaluated for consistent, inconsistent and semiconsistent instances for each problem dimension studied. Figure 5 presents the results of the parallel performance and scalability analysis.

By splitting the population, pCHC has a more focused exploration pattern than the panmictic search. However, the normalized makespan values show that splitting the population in more than 8 demes causes the pCHC results to deteriorate, mainly due to the lose of diversity in every deme. A sample cut of the 3D graphics for a representative dimension (2048×64, including error marks) is presented in Figure 5(d), showing the reduction of the normalized makespan values up to 8 subpopulations, and the slight worse results achieved when using additional subpopulations. Similar results were achieved when solving other HSCP instances. This behavior suggests that there is still work to be done to enhance the pCHC method to achieve a fully scalable scheduler, able to improve over its own results as additional computing resources are available.

7. CONCLUSIONS AND FUTURE WORK

This work has presented a pCHC algorithm for solving the HCSP, a capital problem when executing tasks in distributed HC and grid platforms. In addition, a new set of large-sized HCSP instances was introduced, to model realistic distributed HC and grid scenarios. The test suite comprise several problem instances designed following a methodology based on the well-known ETC model for execution time estimation.

The pCHC was conceived to find accurate HSCP solutions in an efficient way, by using a predefined stopping criterion that allows a quickly planning, and eventually rescheduling of incoming tasks. It was implemented using the MALLBA library and executed in a high performance cluster for solving both a well-known test suite of HCSP instances with reduced size and the new high-dimension instances—specially relevant to analyze the scalability behavior of the proposed method.

The experimental evaluation provided a first step toward a grid-level scalability analysis of pCHC on medium-sized grid scenarios up to 256 processors and scheduling up to 8,192 tasks. PCHC obtained accurate schedules in reduced execution times. It was able to improve over the best previously known makespan results for six instances out of twelve in the problem set by Braun et al., a remarkable efficiency when considering the exhaustive work done by previous researches to solve this small-sized HCSP test suite. When solving the high dimension problem instances, pCHC was also able to achieve significant makespan improvements with respect to the results obtained using traditional (deterministic) list scheduling heuristics. The pCHC obtained makespan improvement factors above 5% over the best deterministic scheduling heuristic for all problem instances. When solving inconsistent and semiconsistent scenarios, the improvements were nearly 10% for all cases, and over 15% for inconsistent instances with dimension 2048×64.

PCHC also demonstrated a high robustness behavior when solving the HCSP, because the standard deviation of the makespan values were very small (below 0.5%) for all problem instances studied. Thus, it can be expected that pCHC will find accurate schedules in any single execution for HCSP scenarios that follow the ETC model by Ali et al. (2000a). The

scalability analysis showed that pCHC has a more focused exploration pattern when using smaller populations. However, the loss of diversity caused the makespan results to deteriorate when splitting the population in more than 8 demes, suggesting that further improvements are needed to achieve a fully scalable scheduler.

The experimental analysis showed that pCHC is a promising method for static scheduling in HCSP scenarios involving thousands of tasks and up to several hundred of machines. However, two main lines remain to be tackled as future work: enhancing the efficacy of the proposed method, and studying the applicability of pCHC to solve dynamic versions of HCSP. The issue related to algorithm efficacy includes studying improved search mechanisms to increase the makespan reductions for consistent scenarios, and also to avoid the lose of diversity that restrained the pCHC scalability to work with no more than eight demes. On the other hand, future work should also focus on studying the applicability of pCHC for solving dynamic versions of HCSP by using a rescheduling strategy that iteratively applies an efficient algorithm for scheduling tasks on dynamic scenarios. In fact, these lines of work are already in progress.

7. ACKNOWLEDGMENTS

The work of S. Nesmachnow and H. Cancela has been partially supported by Programa de Desarrollo de las Ciencias Básicas (PEDECIBA) and Comisión Sectorial de Investigación Científica (CSIC), Universidad de la República, 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

ALBA, E. 2005. Parallel metaheuristics: A new class of algorithms. Wiley: Hoboken, NJ.

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

ALBA, E., F. ALMEIDA, M. BLESA, C. COTTA, M. DIAZ, I. DORTA, J. GABARRÓ, J. GONZÁLEZ, C. LEÓN, L. MORENO, J. PETIT, J. RODA, A. ROJAS, and F. XHAFA. 2002. MALLBA: A library of skeletons for combinatorial optimisation. *In* Proceedings of the Euro-Par, Paderborn, Germany, pp. 927–932.

ALI, S., H. SIEGEL, M. MAHESWARAN, S. ALI, and D. HENSGEN. 2000a. Task execution time modeling for heterogeneous computing systems. *In* Proceedings of the 9th Heterogeneous Computing Workshop, IEEE Computer Society, Washington, DC, pp. 185–200.

ALI, S., H. SIEGEL, M. MAHESWARAN, S. ALI, and D. HENSGEN. 2000b. Representing task and machine heterogeneities for heterogeneous computing systems. Tamkang Journal of Science and Engineering, **3**(3): 195–207.

ANDERSON, T., D. CULLER, D. PATTERSON, and the now team. 1995. A case for now (networks of workstations). IEEE Micro, **15**(1): 54–64.

ARMSTRONG, R., D. HENSGEN, and T. KIDD. 1998. The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions. *In* Proceedings of the Seventh Heterogeneous Computing Workshop, IEEE Computer Society, Washington, DC, pp. 79–83.

BÄCK, T., D. FOGEL, and Z. MICHALEWICZ, editors. 1997. Handbook of evolutionary computation. Oxford University Press: Bristol, UK.

BRASILERO, F., A. DUARTE, D. CARVALHO, R. BARBERA, and D. SCARDACCI. 2008. An approach for the coexistence of service and opportunistic grids: The EELA-2 case. *In* Proceedings of the 2nd Latin-American Grid Workshop, Campo Grande, Brazil.

BRAUN T., H. SIEGEL, N. BECK, L. BÖLÖNI, M. MAHESWARAN, A. REUTHER, J. ROBERTSON, M. THEYS, B. YAO, D. HENSGEN, and R. FREUND. 2001. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel Distributive Computing, **61**(6): 810–837.

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

DAVIS, L. 1991. Handbook of genetic algorithms. van Nostrand Reinhold: New York.

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: San Francisco, CA, pp. 265–283.

FOSTER, I., and C. KESSELMAN. 1998. The grid: Blueprint for a future computing infrastructure. Morgan Kaufmann Publishers: San Francisco, CA.

GAREY, M., and D. JOHNSON. 1979. Computers and intractability. Freeman: New York.

GOLDBERG, D. 1989. Genetic algorithms in search, optimization, and machine learning. Addison Wesley: New York.

HENSGEN, D., T. KIDD, D. St. JOHN, M. SCHNAIDT, H. SIEGEL, T. BRAUN, M. MAHESWARAN, S. ALI, J. KIM, C. IRVINE, T. LEVIN, R. FREUND, M. KUSSOW, M. GODFREY, A. DUMAN, P. CARFF, S. KIDD, V. PRASANNA, P. BHAT, and A. ALHUSAINI. 1999. An overview of MSHN: The management system for heterogeneous networks. *In* Proceedings of the 8th IEEE Workshop on Heterogeneous Computing Systems, San Juan, PR, pp. 184–198.

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

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

LI, H., D. GROEP, J. TEMPLON, and L. WOLTERS. 2004. Predicting job start times on clusters. *In* Proc. of the 2004 IEEE International Symposium on Cluster Computing and the Grid, Computer Society, IEEE, Washington, DC, pp. 301–308.

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

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

NESMACHNOW, S. 2009. Algoritmos evolutivos paralelos para despacho de tareas en entornos heterogéneos. *In* Proceedings of the VI Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, Málaga, Spain, pp. 571–578. (Text in Spanish)

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

RITCHIE, G., and J. LEVINE. 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.

SCHUTTEN, J. 1996. List scheduling revisited. Operations Research Letters, **18**(4): 167–170.

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

XHAFA, F. 2007. A Hybrid Evolutionary Heuristic for Job Scheduling in Computational Grids, chapter 10. Springer Verlag Series: Studies in Computational Intelligence, Vol. 75, pp. 269–311.

XHAFA, F., and B. DURAN. 2008. Parallel Memetic Algorithms for independent job scheduling in computational grids. *In* Recent Advances in Evolutionary Computation for Combinatorial Optimization, Vol. 153. Studies in Computational Intelligence, Springer, pp. 219–239.

XHAFA, F., E. ALBA, and B. DORRONSORO. 2007a. Efficient batch job scheduling in grids using cellular memetic algorithms. *In* Proceedings of the 21st International Parallel and Distributed Processing Simposium. IEEE Computer Society, Long Beach, CA, pp. 1–8.

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

XHAFA, F., J. CARRETERO, E. ALBA, and B. DORRONSORO. 2008a. Design and evaluation of tabu search method for job scheduling in distributed environments. *In* Proceedings of the 21st International Parallel and Distributed Processing Symposium, IEEE, Shanghai, China, pp. 1–8.

XHAFA, F., B. DURAN, A. ABRAHAM, and K. DAHAL. 2008b. Tuning struggle strategy in genetic algorithms for scheduling in computational grids. *In* Proceedings of the 7th Computer Information Systems and Industrial Management Applications, IEEE Computer Society, Washington, DC, pp. 275–280.

YANG, J., I. AHMAD, and A. GHAFOOR. 1993. Estimation of execution times on heterogeneous supercomputer architectures. *In* Proceedings of the 1993 International Conference on Parallel Processing, IEEE Computer Society, Washington, DC, pp. 219–226.

ZOMAYA, A., and Y. TEH. 2001. Observations on using genetic algorithms for dynamic load-balancing. IEEE Transactions on Parallel Distributive System, **12**(9): 899–911.

APPENDIX: BEST SOLUTIONS FOR THE TEST SUITE FROM BRAUN ET AL.

This appendix reports the best solutions obtained by pCHC for six problems on the test suite by Braun et al. (2001). For the sake of simplicity, the solutions are presented in a task-oriented representation (array of machine identifiers, the presence of m_j in the position t_i means that the task t_i is scheduled to execute on machine m_j, see Section 5.2.1).

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

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

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

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

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

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