Future Generation Computer Systems 60 (2016) 48–66

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Scheduling independent tasks on heterogeneous processors using heuristics and Column Pricing

Christos Gogos [a,*], Christos Valouxis [b], Panayiotis Alefragis [c], George Goulas [c], Nikolaos Voros [c], Efthymios Housos [d]

[a] Technological Educational Institute of Epirus, Department of Computer and Informatics Engineering, Arta, Greece
[b] Ministry of Education, Achaia Branch Office, Patras, Greece
[c] Technological Educational Institute of Western Greece, Department of Computer and Informatics Engineering, Patras, Greece
[d] University of Patras, Department of Electrical and Computer Engineering, Patras, Greece

ARTICLE INFO

Article history:
Received 30 May 2015
Received in revised form
28 December 2015
Accepted 24 January 2016
Available online 3 February 2016

Keywords:
Heterogeneous processors
Independent tasks
Task scheduling
Heuristics
Mathematical programming
Column pricing

ABSTRACT

Efficiently scheduling a set of independent tasks on a virtual supercomputer formed by many heterogeneous components has great practical importance, since such systems are commonly used nowadays. Scheduling efficiency can be seen as the problem of minimizing the overall execution time (makespan) of the set of tasks under question. This problem is known to be NP-hard and is currently addressed using heuristics, evolutionary algorithms and other optimization methods. In this paper, firstly, two novel fast executing heuristics, called LSufferage and TPB, are introduced. L(ist)Sufferage is based on the known heuristic Sufferage and can achieve in general better results than it for most of the cases. T(enacious)PB is also based on another heuristic (Penalty Based) and incorporates new ideas that significantly improve the quality of the resulted schedule. Secondly, a mathematical model of the problem is presented alongside with an associated approach based on the Linear Programming method of Column Pricing. This approach, which is called Column Pricing with Restarts (CPR), can be categorized as a hybrid mathematical programming and heuristic approach and is capable of solving in reasonable time problem instances of practically any size. Experiments show that CPR achieves superior results improving over published results on problem instances of various sizes. Moreover, hardware requirements of CPR are minimal.

1. Introduction

During the last decades technology progress transformed computing power from a property of the few to an asset that can be used by virtually anybody. Homogeneous systems, which refers to multiprocessor systems consisting of identical processing units, were the first wave of systems that was build in order to serve as a processing infrastructure capable of parallel execution for a set of tasks. These tasks typically have execution dependencies among them captured in task graphs. The second wave was heterogeneous systems, consisting of several interconnected computers without common hardware characteristics running under a resource coordination software. Grid computing [1] has been established as the general term when referring to such systems. Under the promise that in a cost effective manner they can execute multiple processing tasks of varying complexity, they have become extremely popular. The problem of task scheduling is critical in homogeneous systems and grid computing [2], since substandard task-processor allocations result to performance losses. For the case of homogeneous systems, heuristics and techniques mainly originated from Operations Research were proposed in order to address the problem of scheduling [3–5]. Likewise, several techniques were proposed for the scheduling problem on grid computing systems [6–8].

In this paper the scheduling problem addressed is the Heterogeneous Computing Scheduling Problem (HCSP), which refers to independent tasks that should be assigned to processors of various characteristics under the goal of minimizing the latest task's finish time. This value is known as makespan and is usually used as the performance metric for scheduling problems. Scheduling problems are in general NP-Hard [9] and this fact is commonly used as a justification that exact methods cannot be applied on solving them when problems of considerable size should be addressed.

* Corresponding author.
E-mail address: cgogos@teiep.gr (C. Gogos).

http://dx.doi.org/10.1016/j.future.2016.01.016
0167-739X/

Fortunately, this is not the case for all scheduling problems, since in several cases, problem specific idiosyncrasies can be exploited, pushing up the limit of problem sizes that can be solved using exact methods. When problem sizes that can be solved using exact methods equals or exceeds the size of problems with practical importance these methods tend to have a solution quality advantage over various approximate methods. This seems to be the case with HCSP and the method CPR that is proposed in Section 5 of this manuscript.

Makespan might be the most common objective used but it has to be mentioned that a number of other objectives also exist such as resource utilization, flow-time and matching proximity [10]. Resource utilization measures the degree of utilization of the resources. Flow-time is the sum of the finishing times of the tasks and can be seen as an indicator of the Quality of Service (QoS) of the system. Matching proximity measures the proximity of the schedule to the schedule that assigns each task to the processor that can execute it fastest. Each one of the aforementioned objectives might be optimized independently or a multiobjective problem might be formulated that targets the simultaneous optimization of more than one objectives [11,12]. Nevertheless, the majority of published work on task scheduling regards makespan as the most important performance metric and uses it apart from other metrics for comparisons among produced schedules.

This manuscript is organized as follows. The next section introduces the problem, presents related work and describes in detail the heuristics Sufferage and Penalty Based (PB) since they form the base for the novel heuristics LSufferage and TPB that are presented next. It is argued that LSufferage and TPB can be used as alternatives to already established heuristics for the problem. Next, a mathematical model of the problem and an associated Column Pricing approach is presented. The Column Pricing approach combined with some simple heuristics is capable to solve problem instances of large sizes that would have been impractical to address directly with an Integer Programming solver. The next section presents experiments that demonstrate the efficiency of the two new heuristics and the Column Pricing approach across a large number of known problem instances. Finally, conclusions of the research alongside with future directions are presented.

2. Problem description

The problem of assigning tasks to processors and specifying the order of execution on each processor is referred to as mapping. In the problem setting of this work tasks are considered to be independent, meaning that the completion of execution of a task is not a precondition for the execution of another task. The set of tasks is usually called meta-task and the goal is to minimize the makespan of the meta-task. Mapping of the meta-task occurs before the start of the meta-task execution, so the problem can be categorized as static scheduling. Furthermore, once a task is scheduled on a processor this task has to finish its execution on it, so pre-emptive execution is prohibited. A related problem that can be found in the bibliography is the Directed Acyclic Graph (DAG) scheduling problem. In contrast with HCSP, the DAG scheduling problem assumes a DAG that captures the precedence constraints and the communication costs that incur when a task that is scheduled to a processor has to transfer data to another task that is supposed to be scheduled to another processor [13]. Meta-tasks occur in several real life situations. For example a set of individual jobs submitted independently for execution to a grid computing is a meta-task.

2.1. The expected time to compute model

In order to simulate different Heterogeneous Computing environments and meta-task characteristics the Expected Time to Compute (ETC) model was used. ETC model was introduced by Ali et al. [14] and defines three metrics: task heterogeneity, processor heterogeneity and consistency type. Task heterogeneity refers to the variation among the execution times of tasks for a given processor. Processor heterogeneity refers to the variation of execution times for a single task across all the processors. Both of them can be either high or low. The third metric, consistency, assumes three values: consistent, inconsistent or partially consistent. When a dataset is consistent this means that if a processor executes a task faster than another processor, then it should execute all other tasks faster than the other processors. If the previous assertion does not hold, then the dataset is considered to be inconsistent except for the cases of semi-consistent datasets where a consistent subset of processors can be identified among them. The expected execution times of the tasks are arranged in a matrix called ETC, where the entry $ETC[t, p]$ is the expected time to compute task t on processor p assuming that the time needed to move the executables and the associated data is also included. These values are supposed to be known a priori while approaches that can be used in order to extract them include task profiling and analytical benchmarking. Further details about the ETC model alongside with alternative computational models for Grid scheduling can be consulted in [10]. Twelve instances of the ETC model involving 512 tasks and 16 processors were generated in [15] and since then used in several papers. Furthermore, 96 much larger problem instances of up to 8192 tasks and 256 processors, were introduced in [16]. So, a testbed of public problem instances and associated published results exist. This testbed was used in order to assert the performance of the approaches proposed later in this paper.

2.2. Related work

A non exhaustive list of approaches to HCSP that our research found includes heuristics [15,17], local search [18,19], Simulated Annealing [20], Genetic Algorithms (GAs) [21–23], Memetic Algorithms (MAs) [24,25], Ant Colony Optimization (ACO) [26,27] and Bee Colony Optimization (BCO) [28]. Among all approaches found, a parallel version of GA called pμ-CHC [16], running on a four QuadCore server cluster, reached very good makespan values close to the theoretical optimums for the same problem instances that were mentioned in the previous paragraph.

Heuristics might not give optimal results but are often used in practice due to their low computational cost and ease of implementation. Several of them are referenced in the bibliography [29] and one way to categorize them is according to the moment that they make the decision of scheduling. The resulting categories are online and batch mode. Online heuristics schedule tasks immediately upon arrival. For example MCT (Minimum Completion Time) heuristic assigns each task to the processor that will result to the earliest completion time for it. On the other hand, batch mode heuristics schedule each task by exploiting the execution times known about the other tasks. The following batch mode heuristics: Min–Min, Min–Max, RC (Relative Cost), PB (Penalty Based) and Sufferage, will be presented next. In all of the algorithms that follow, T is the set of tasks, P is the set of processors, $\text{ETC}(t, p)$ is the estimated time to complete task $t \in T$ on processor $p \in P$ and $\text{EFT}(t, p)$ is the earliest finish time of task t on processor p given the current state of assignments that have already been set up to the point of scheduling task t.

2.2.1. The Min–Min algorithm

Min–Min algorithm is a simple scheduling algorithm that is reported to consistently produce good results. In each step of Min–Min a single task is scheduled. This occurs by computing the earliest finish time (EFT) over all processors of all still unscheduled tasks in order to schedule the task to the processor that has the minimum EFT value over all other alternatives. Min–Min

was proposed back in 1977 [30] and despite its age it is still commonly used as the base algorithm for comparisons with other heuristics. There are references in the literature [31,32] that report Min–Min to have runtime complexity of $O(PT^2)$. Nevertheless, a clever implementation of the algorithm, that switches from the original task oriented view to a processor oriented view, as described in [33], manages to lower the runtime of Min–Min to $O(PT \log T)$. Furthermore, using radix exchange sort which is a non comparison sort algorithm lowers the time complexity of Min–Min to $O(PT)$ [34].

2.2.2. The Min–Max algorithm

Like Min–Min, the Min–Max algorithm [35] schedules one task at a time until no more tasks are left. At each step, the minimum completion times of all unassigned tasks over all available processors are computed. Then, for each unassigned task, the ratio of its minimum execution time (execution time on the fastest processor for this task) to the execution time on the processor that resulted to the minimum completion time is computed. The task that has the highest value of this ratio is scheduled and removed from the list of unassigned tasks. The choices that Min–Max makes are justified by the fact that higher values for the ratio mean that on each step the selected task will be scheduled to the processor that can execute it in time relatively close to the minimum possible execution time. The runtime complexity of Min–Max is $O(PT^2)$.

2.2.3. The relative cost algorithm

This algorithm utilizes an indicator called Relative Cost (RC) that incorporates the notion of load balance that is overlooked by other heuristics like Min–Min. Two quantities are computed for each task and processor combination: the static relative cost and the dynamic relative cost. The static relative cost is computed once at the start of the algorithm as $\gamma_s(t,p) = ETC(t,p)/ETC(t)_{avg}$ where $ETC(t)_{avg}$ is the average execution time of task t over all available processors. The dynamic relative cost is computed before each task is scheduled as $\gamma_d(t,p) = EFT(t,p)/EF(t)_{avg}$, where $EFT(t,p)$ is the finish time of task t on processor p and $EF(t)_{avg}$ is the average finish time of task t over all available processors. The task and processor that are selected are the ones that minimize the expression $\gamma_d^{\alpha} \times \gamma_s$. Typical value used for α is 0.5. More details about the RC algorithm can be found in [36]. The runtime complexity of RC is $O(PT^2)$ as in classic Min–Min and Min–Max but it is expected to perform slower than them due to its intricacy.

2.2.4. The sufferage algorithm

Sufferage is an algorithm based on the concept that a task "suffers" when it is not scheduled to the processor that could execute it faster than all the other processors [37]. The value of suffer (sufferage) for each task is defined to be the difference between the second best and the best processor execution time for the specific task. Once sufferage values are determined for all tasks the task with the maximum sufferage value is assigned to the processor that has the minimum completion time for that task. At each round, at most $|P|$ tasks are scheduled to the most favorable processor for them according to their sufferage values. Algorithm 1 presents the associated pseudo-code. The runtime complexity of Sufferage is $O(PT^2)$.

Algorithm 1 The Sufferage algorithm

```
1: while T ≠ ∅ do
2:     Mark all p ∈ P as unassigned for the current round
3:     T_round ← T
4:     while T_round ≠ ∅ do      ▷ each t ∈ T_round is considered only once
5:         Select t ∈ T_round
6:         min_1 ← +∞
7:         min_2 ← +∞
8:         for all p ∈ P do
9:             if EFT(t, p) < min_1 then
10:                min_2 ← min_1
11:                min_1 ← EFT(t, p)
12:                pmin_1 ← p
13:            else if EFT(t, p) < min_2 then
14:                min_2 ← EFT(t, p)
15:            end if
16:        end for
17:        sufferage ← min_2 − min_1
18:        if pmin_1 is unassigned then
19:            Assign t to pmin_1
20:            Mark pmin_1 as assigned for the current round
21:        else if sufferage > sufferage of t′ already assigned to pmin_1 then
22:            Unassign t′ from pmin_1
23:            Assign t to pmin_1
24:            T ← T ∪ t′
25:        end if
26:        T_round ← T_round \ t
27:    end while
28:    Schedule t ∈ T_round according to assignments made in this round
29:    T ← T \ T_round
30: end while
```

2.2.5. The penalty based algorithm

The Penalty Based (PB) algorithm [38] is a two stage heuristic. In the first stage, each task is assigned to the processor that can execute it faster. This causes imbalance, especially for the consistent type of problems. Then, at each step of the second stage, a task is transferred from the maximally loaded processor A to the minimally loaded processor B provided that the new completion time should be less than the average of the previous completion times of A and B. The task that should be moved at each step of the second stage is the one with the minimum penalty, where penalty is defined to be the difference between the execution time of t in B from the execution time of t in A divided by the execution time of t in A. The penalty represents the ratio that processor A can execute a certain task faster than processor B, so higher penalty values mean that a task is moved further away from the processor that could execute it faster. Algorithm 2 presents the associated pseudo-code. The runtime complexity of PB is $O(TP + T^2)$, where the TP part of the expression corresponds to the initial assignment of each task to the fastest processor for it and the rest of the expression corresponds to the two nested loops over tasks that in the worst case will be iterated T times each. In practice, PB as implemented in Algorithm 2, proves to be faster than all the other algorithms described in this manuscript.

3. New heuristics

Two new heuristics are proposed: LSufferage and TPB. Both of them are inspired by existing heuristics and they have very good performance and execution time characteristics.

3.1. The list sufferage algorithm

LSufferage is a new heuristic proposed here that exploits the concept of Sufferage combined with the generation of static ordered lists of tasks that speeds up the algorithm's execution. Pseudo-code of the algorithm is presented in Algorithms 3 and 4.

Algorithm 2 The Penalty-Based algorithm

```
1: Assign each t ∈ T to the processor that can execute it faster
2: repetitions ← 0
3: flag ← true
4: while repetitions < |T| AND flag do
5:     repetitions ← repetitions + 1
6:     selectedTask ← −1
7:     minpenalty ← +∞
8:     pa ← processor that is maximally loaded
9:     pb ← processor that is minimally loaded
10:    ca ← completion time of pa
11:    cb ← completion time of pb
12:    for all t ∈ tasks scheduled in pa do
13:        if cb+ETC(t, pb) < (ca+cb)/2 then
14:            penalty ← (ETC(t,pb)−ETC(t,pa))/ETC(t,pa)
15:            if penalty < minpenalty then
16:                minpenalty ← penalty
17:                selectedTask ← t
18:            end if
19:        end if
20:    end for
21:    if selectedTask ≠ −1 then
22:        Transfer the selectedTask from pa to pb
23:    else
24:        flag ← false
25:    end if
26: end while
```

For each processor p, a size $|T|$ list of pairs is constructed. Each one of these lists has exactly one pair for every task $t \in T$ which contains the task number and a value representing the priority of executing task t on processor p which is defined in Eq. (1). In this equation p_t^1 and p_t^2 are the processors that execute task t fastest and second fastest respectively.

$$\text{priority of } t \text{ in } p = \begin{cases} \dfrac{ECT[t, p_t^2]}{ECT[t, p_t^1]} & \text{if } p = p_t^1 \\ \dfrac{ECT[t, p_t^1]}{ECT[t, p]} & \text{otherwise.} \end{cases} \quad (1)$$

By sorting each list according to priority values, an order of preferred task execution for each processor is formed. For each task t that is executed fastest on processor p, the priority value becomes the ratio of the execution time of task t on the second fastest processor for t over the execution time of task t on p. So, it is preferred to grand priority of execution to these tasks that if they miss the opportunity this would result to higher cost overheads compared to other tasks that are also executed faster on the same processor. This way of computing priorities is similar to the way of computing"suffer" value on the Sufferage heuristic presented at Section 2.2.4. All these task priority values are guaranteed to be greater than or equal to one. On the other hand, if task t does not have processor p as the fastest processor, the priority value of task t on processor p becomes the ratio of the execution time of task t on the fastest processor for t over the execution time of task t on processor p. Therefore, among tasks that could execute faster on a processor other than p, tasks that have a smaller difference of execution times compared to the execution time on the best possible for them processor are given priority. These priority values have values less than or equal to one, so tasks associated with them are all given lower priority of execution on processor p than other tasks that have processor p as the fastest possible processor. Each list gets sorted in descending order according to priority values and the corresponding task numbers becomes a static priority of the execution list for processor p. The main idea of improving the performance of heuristics like Min–Min and Sufferage is keeping for each task the Minimum Completion Time values that are associated with each processor separately maintained. This idea is also presented in [34,33] and the resulting runtime complexity of LSufferage is therefore $O(PT \log T)$.

Algorithm 3 The LSufferage algorithm

```
1:  ▷ min1 and min2 are vectors of size |T| containing the time needed by task t to execute on the best and on the 2nd best possible processor for t
2:  for all p ∈ P do
3:      priorityValues ← [ ]   ▷ ∀p, priorityValues is a list of (task, priority) pairs that defines the priority of execution among tasks
4:      for all t ∈ T do
5:          if p is the fastest processor for t then
6:              priority ← min2[t]/min1[t]   ▷ (values ≥ 1)
7:          else
8:              priority ← min1[t]/ETC(t, p)   ▷ (values ≤ 1)
9:          end if
10:         priorityValues.add(t, priority)
11:     end for
12:     sortedTasksPerProcessor[p] ← sort(priorityValues)   ▷ Sort priorityValues in descending order according to priority
13: end for
14: scheduleBasedOnStaticOrder(sortedTasksPerProcessor)
```

Algorithm 4 The scheduleBasedOnStaticOrder procedure

```
1:  procedure scheduleBasedOnStaticOrder(sortedTasksPerProcessor)
2:      Initialize sortedTasksIndices with zero values
3:      for all t ∈ T do
4:          for all p ∈ P do
5:              update(sortedTasksIndices[p], sortedTasksPerProcessor)   ▷ update sortedTasksIndices[p] to the column index of the next yet to be scheduled task in sortedTasksPerProcessor[p]
6:          end for
7:          min ← +∞
8:          for all p ∈ P do
9:              t′ ← sortedTasksPerProcessor[p][sortedTaskIndices[p]]
10:             if EFT(t′, p) < min then
11:                 min ← EFT(t′, p)
12:                 tmin ← t′
13:                 pmin ← p
14:             end if
15:         end for
16:         Schedule tmin to pmin
17:     end for
18: end procedure
```

3.2. The tenacious penalty based algorithm

The TPB heuristic is based on the PB heuristic. It extends the idea of moving a task from the maximally loaded processor A to the minimally loaded processor B, to moving a task from A to any other processor by checking processors in ascending order of load. So, in each repetition, the processors are sorted according to their current loads. In order for a task to be eligible for transfer from processor A to another processor P the updated completion time of P has to be less than the previous completion time of A, which is another point of difference with the PB heuristic that used the average of completion times of the two processors. The runtime complexity of TPB is $O(PT^2)$.

By comparing Algorithm 2(PB) with Algorithm 5(TPB) it can be observed that TPB is computationally heavier than PB. In each repetition processors are sorted according to their load. An extra

Algorithm 5 The Tenacious Penalty Based algorithm

```
Assign each t ∈ T to the processor that can execute it faster
repetitions ← 0
while repetitions < |T| do
    repetitions ← repetitions + 1
    selectedTask ← −1
    minpenalty ← +∞
    pl ← Sort in ascending order the processors based on their load
    pa ← last element of pl
    ca ← completion time of pa
    for all pb ∈ pl excluding pa do
        cb ← completion time of pb
        for all t ∈ tasks scheduled in pa do
            if cb+ETC(t, pb) < ca then
                penalty ← (ETC(t,pb) − ETC(t,pa)) / ETC(t,pa)
                if penalty < minpenalty then
                    minpenalty ← penalty
                    selectedTask ← t
                    selectedProcessor ← pb
                end if
            end if
        end for
        if selectedTask ≠ −1 then
            Transfer selectedTask from pa to selectedProcessor
            Continue the while loop
        end if
    end for
    if selectedTask = −1 then
        return
    end if
end while
```

loop is needed in comparison with PB since not only the minimally loaded processor is tested to be the target of each task transfer but possibly all the available processors. In practice, since a processor that can be used to offload the maximum loaded processor is usually found early in the process, the algorithm keeps to be fairly fast.

4. A mathematical model for the HCSP

It is common to express a problem using a mathematical model in order to formally capture it. Here, the problem is portrayed mathematically not only for better understanding, but also as a mean to achieve better results and to identify lower bounds that exist for the solution of the problem examined. Moreover, a relaxed version of this model will be used in the Column Pricing approach that follows in Section 5.

The mathematical model of the problem is fairly simple and compact since its objective function consists of a single variable, it has only one set of decision variables, and two sets of constraints. Decision variables of the problem are the binary variables y_{tp} which are defined over each $t \in T$ and $p \in P$. Variable y_{tp} assumes value 1 if task t is scheduled to processor p or 0 otherwise.

The first set of constraints ensures that each task will be assigned to exactly one processor.

$$\sum_{p \in P} y_{tp} = 1 \quad \forall t \in T. \tag{2}$$

The second set of constraints enforces variable m to assume the maximum value among execution times over all processors. Given that the execution time of each task on each processor is stored in parameter w_{tp}, the right side of inequality (3) assumes the value of the total execution time of all tasks on processor p.

$$m \geq \sum_{t \in T} w_{tp} y_{tp} \quad \forall p \in P. \tag{3}$$

The objective function of the model captures the fact that the completion time of the schedule should be minimized. Since the objective function is minimized, this pushes the value of m to assume the smallest possible time that the largest of execution times is completed. This value is exactly the makespan of the generated schedule.

$$\text{minimize } m. \tag{4}$$

Although simple, the mathematical model is operational and can be used in practice in order to solve efficiently problem instances of up to a certain size. In order to test the mathematical model, a solver called thereafter "Math-Solver" was implemented in Java using the Gurobi 6.0.3 Integer Programming (IP) solver [39] with the default configuration. Math Solver was able to produce high quality solutions for the Braun et al. (512 tasks and 16 processors) problem instances but for larger problem instances failed to directly address them. This was expected since in the 1024 tasks and 32 processors class of datasets, the resulting IP model had 32 769 binary variables (columns) and 1056 constraints (rows) while for the 8192 tasks and 256 processors class of datasets, the resulting IP model had 2 097 153 columns and 8448 rows. Such problem sizes are difficult to be addressed even by state of the art IP solvers such as Gurobi. Nevertheless, the mathematical model is still useful for larger problems provided that decomposition techniques are used. Such an approach will be presented in the following section.

5. The column pricing approach

Here, an approach capable of giving near optimal solutions to HCSP problem instances of big sizes is described. A high level description of the approach is given in Algorithm 6.

Algorithm 6 The Column Pricing approach

```
Construct an initial solution using a heuristic
bestIPSolution ← initial solution
Solve LP including variables y_tp that exist in the solution
Compute reduced costs ∀y_tp ∉ LP BASIS
while ∃y_tp with negative reduced cost do
    Formulate new LP keeping y_tp in LP BASIS and adding promising y_tp variables
    Solve the LP
    Compute reduced costs ∀y_tp ∉ LP BASIS
    Transform fractional solution to integer solution
    Apply heuristics to improve the integer solution
    Replace bestIPSolution with new IP solution if better
end while
```

Initially, a starting solution of good quality is generated using the heuristic algorithm TPB. This solution determines which y_{tp} variables will be included in the mathematical model. So, when the heuristic solution has task t' scheduled to processor p', the variable $y_{t'p'}$ is included in the mathematical model of the problem to be solved. So, the initial formulation of the mathematical model has only one variable y_{tp} for each task t. In order to solve the mathematical model using a Linear Programming solver the problem is relaxed by removing the integrality constraint previously imposed over the decision variables. The first time that the problem is solved the solution will be an integer one, the

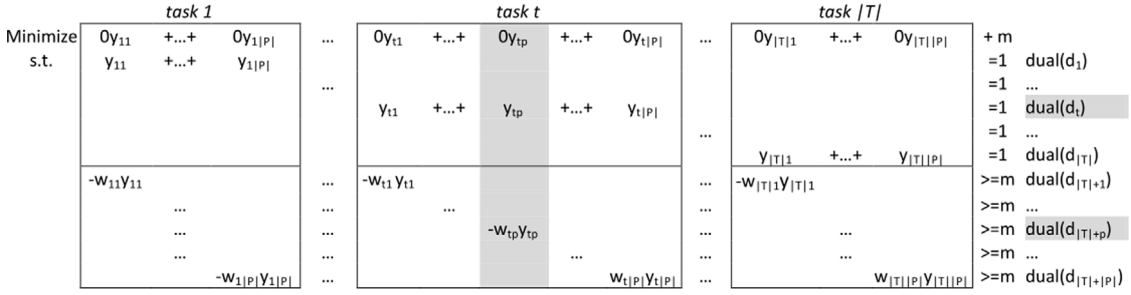

Fig. 1. Values involved in computing the reduced cost of variable y_{tp}.

objective cost will be equal to the cost of the initial solution and more importantly dual values will be produced for each constraint of the model. The dual values will then be used in order to compute reduced costs for each variable y_{tp} not included in the LP Basis. This is done by using the formula presented in Eq. (5), where d_t are the dual values for the first $|T|$ constraints and $d_{|T|+p}$ are the dual values for the next $|P|$ constraints as described in Section 4 for the first and the second set of constraints respectively. Fig. 1 shows the positions of the corresponding coefficients and dual values in the formulation of the problem.

$$rc_{y_{tp}} = 0 - d_t + w_{tp} d_{|T|+p} \quad \forall t \in T,\ p \in P. \tag{5}$$

Then, a number of "promising" y_{tp} variables are selected based on their reduced costs so as to be included in the formulation of the next mathematical problem which should already include all the basic variables of the previous problem. Since the reduced cost of a non basic variable is the amount that the objective function will be changed if the variable becomes basic, reduced cost values with more negative values should be preferable when a minimization problem is considered. In this problem setting, a great number of y_{tp} variables are expected to have negative values, specially in the earlier stages of the method. A certain number of them have to be selected and in order to have a fair amount of y_{tp} variables for each task t, an upper bound of 20 variables with more negative values are allowed for each task. Furthermore, the total number of y_{tp} variables should be less than 25 000 since experiments showed that the Linear Programming solver is able to solve problems of this size in time less than a second, which is needed since several of such problems will be solved in the process. In case bigger problems than the ones examined in this paper had to be addressed, the total allowed number of y_{tp} variables would have to be increased. Since LP solvers are capable of solving LPs with hundreds of thousands of variables that could be possible but of course longer runtimes would have to be tolerated then.

Next, the Linear Programming solver solves the problem and a new solution is generated, most certainly having fractional values for the decision variables involved. Since we are interested in an integer solution the current solution is transformed to an integer one. Certain variables belonging to the LP basis have to assume value one given that each and every task are scheduled to a single processor. This occurs by computing for each task t the average of the minimum and maximum values of all basic variables associated with it and then selecting randomly one of them that has value not less than the average. For example, if in the LP solution, task t has three basic variables y_{tp_1}, y_{tp_2} and y_{tp_3} with fractional values 0.4, 0.1 and 0.5 respectively, then the average of minimum and maximum will be 0.3 and the variable that will assume value one will be randomly selected between y_{tp_1} and y_{tp_3}, ignoring variable y_{tp_2}. Using this technique, basic variables having relatively low fractional values are discarded which seems to be a good strategy. By selecting one basic variable for each task to assume value 1, an integer solution emerge.

Then, four heuristics which can be described as local searches try to improve the quality of it. All of them involve tasks selected from the maximally loaded processor, called $p_{\max}$ hereafter. Since $p_{\max}$ is the processor that terminates execution last, it defines the quality of the solution and plays a central role at the heuristics that follow. It is possible that by moving a task from $p_{\max}$ it is no longer the processor that is maximally loaded. So, after moving a task a possibly new $p_{\max}$ is identified and each heuristic continues for the tasks that the new $p_{\max}$ hosts. Furthermore, in order to avoid multiple moves of the same task, once a task is moved it is tabooed and no longer participates in further moves initiated by the same heuristic. The purpose of using four iterative improvement heuristics was to allow better exploration of the problem domain space through local search. Our experiments showed that when some improvement heuristics were deactivated the returned results were less good than when all four of them were in use.

5.1. Heuristic 1

The first improvement heuristic tries to move single tasks from $p_{\max}$ to another processor. Each task t' of $p_{\max}$ is tested whether by moving it to another processor lowers the makespan. If this is the case then t' is moved. Similar to the TPB heuristic described in Section 3.2, processors are sorted in ascending order of task's t' execution time and then examined in turn until a successful move is found or no such move exists. The process continues until no more moves of tasks hosted by the current $p_{\max}$ can be identified.

5.2. Heuristic 2

The second improvement heuristic tries for each task of $p_{\max}$ to swap it with another task from some other processor. All possible pairs formed by a task of $p_{\max}$ and a task from another processor are examined and when such a pair results to lower makespan, the tasks swap processors. The heuristic continues until no more pairs of this kind can be found.

5.3. Heuristic 3

The third improvement heuristic tries to find a task from $p_{\max}$ that can be moved to another processor from which another task will be moved to the processor that can execute it fastest. This sequence of moves have to result to lower makespan in order to be accepted. The heuristic continues until no such moves can be found.

5.4. Heuristic 4

Like the previous heuristics, the last one tries for each task of $p_{\max}$ to move it to another processor. From this processor another

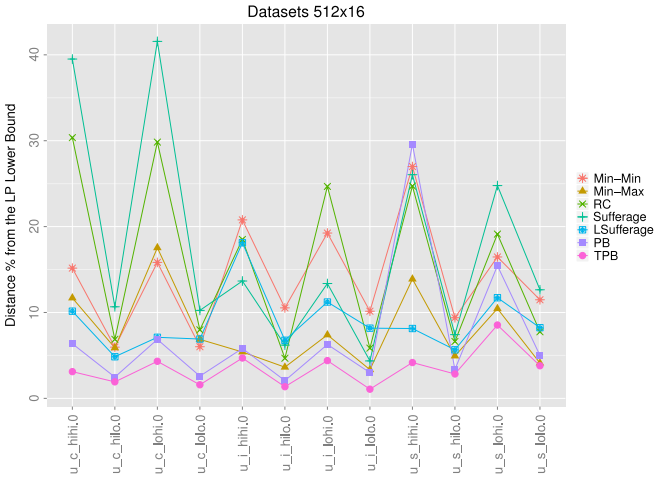

Fig. 2. LSufferage(LS) and TPB vs. other heuristics (512 × 16).

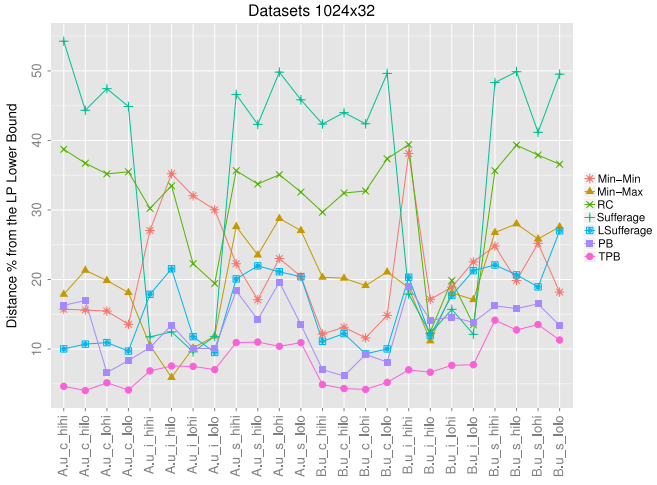

Fig. 3. LSufferage and TPB vs. other heuristics (1024 × 32).

task t' then have to be transfered to the processor that is the minimally loaded among processors.

Each one of the improvement heuristics is activated in turn resulting in a new integer solution for the problem. If it is better than the previous best integer solution it is stored and the process continues by solving the next LP problem. If no variables y_{tp} with negative reduced costs can be found, then a lower bound for the LP problem has been reached. Then two options exist. Either the process ends returning the best IP solution found or provided that more time is available the process restarts using a new initial solution. This new initial solution has 90% of assignments copied from the solution of the previous iteration while the rest of the assignments are assigned randomly. For the experiments that follow the first approach that ends when the LP lower bound is found will be called Column Pricing Downhill (CPD) while the second approach that restarts whenever time is available will be called Column Pricing with Restarts (CPR).

6. Experiments

Experiments were performed against the 12 "classic" datasets proposed by Braun et al. [15] and the 96 much larger problem instances introduced by Nesmachnow et al. [16]. The former datasets referred to as "0.0" in the original article consist of 12 problems instances with each one having 512 tasks and 16 processors. They have been used across many papers since they became available.

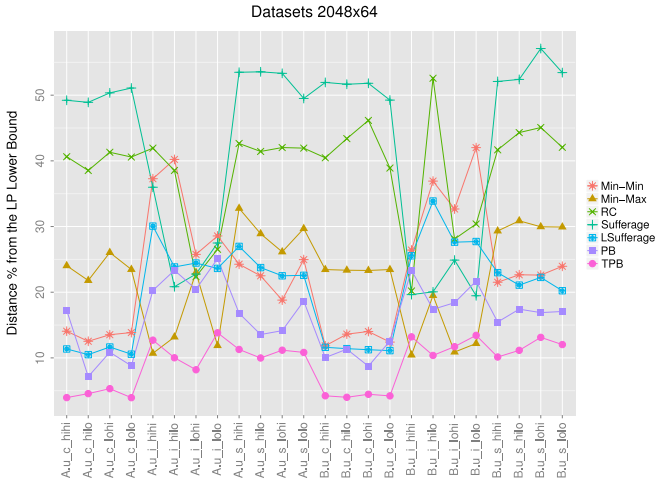

Fig. 4. LSufferage and TPB vs. other heuristics (2048 × 64).

The latter datasets are publicly available to download at the HCSP website http://www.fing.edu.uy/inco/grupos/cecal/hpc/HCSP/ and cover the following scenarios: 1024 tasks and 32 processors, 2048 tasks and 64 processors, 4192 tasks and 128 processors and finally 8192 tasks and 256 processors.

The names of all problem instances used in the experiments share a common pattern which is *d_c_MT*, where *d* shows the distribution function used to generate the values, *c* indicates the consistency type, and *M* and *T* indicate the heterogeneity for the machines and the tasks respectively. In all the problem instances *d* assumes value *u* for uniform distribution, *c* assumes value *c* or *i* or *s* for consistent, inconsistent and semiconsistent and *M* and *T* assumes either value *hi* for high heterogeneity or *lo* for low heterogeneity. The meaning of consistency and heterogeneity were described in Section 2.1. All experiments presented hereafter were run on a Windows 7 64 bit computer equipped with an Intel Core i7 860 2.8 GHz processor and 16 GB RAM memory. All algorithms were implemented in Java 8 (version 1.8.0_60 64 bit).

6.1. Experiments over selected heuristics

Tables 1–5 present the results that Min–Min, Min–Max, RC, Sufferage, LSufferage, PB and TPB produced. The last column of each table presents the lower bound for each problem instance as computed by the Linear Programming relaxation of the mathematical model given in Section 4. Next to the value scored by each heuristic is its execution time in parentheses. The execution time is calculated by averaging the execution time of 50 runs for datasets 512 × 16, 1024 × 32 and 2048 × 64, 20 runs for datasets 4096 × 128 and 10 runs for datasets 8192 × 256. The last line of each table presents the average execution time achieved by each heuristic over all problem instances included in the corresponding datasets. Since datasets 512 × 16 have very short execution times, their execution times are presented in milliseconds (ms), while in all other datasets execution times are in seconds.

Moreover, Figs. 2–6 show for each problem instance how each heuristic performs. In particular, the LP lower bound assumes the base line position and the heuristics are displayed as points higher according to the percentage that their value is worse than the lower bound. So, points that have higher positions at the figures correspond to heuristics that fail to produce good values.

By observing the tables and figures produced by the experiments it can be noticed that both of the new proposed heuristics LSufferage and TPB outperform the heuristics that they were inspired from, Sufferage and PB respectively. This occurs in 81 out of

Table 1
Results achieved by selected heuristics on Braun et al. (512 × 16) datasets.

Dataset	Min–Min (ms)	Min–Max (ms)	RC (ms)	Sufferage (ms)	LSufferage (ms)	PB (ms)	TPB (ms)	LP bound
u_c_hihi.0	8 460 675.0 (1.51)	8 205 561.3 (6.14)	9 576 839.0 (25.50)	10 249 172.9 (7.63)	8 092 234.8 (1.34)	7 813 994.3 (1.25)	7 575 121.1 (1.46)	7 346 524.2
u_c_hilo.0	161 805.4 (1.34)	161 686.8 (5.78)	163 200.2 (25.20)	168 982.6 (7.68)	160 100.3 (1.27)	156 482.5 (1.29)	155 627.1 (1.60)	152 700.4
u_c_lohi.0	275 837.4 (1.25)	279 907.7 (6.18)	309 192.7 (25.50)	337 121.5 (7.53)	255 070.3 (1.18)	254 485.1 (1.06)	248 411.5 (1.34)	238 138.1
u_c_lolo.0	5441.4 (1.27)	5485.4 (5.69)	5542.6 (25.68)	5658.5 (7.61)	5487.4 (1.27)	5265.6 (1.26)	5213.8 (1.53)	5132.8
u_i_hihi.0	3 513 919.3 (1.34)	3 066 454.8 (6.89)	3 447 651.4 (25.98)	3 306 818.9 (10.40)	3 436 518.1 (1.33)	3 078 540.2 (0.10)	3 046 010.7 (0.92)	2 909 326.6
u_i_hilo.0	80 755.7 (1.41)	75 711.6 (6.48)	76 471.5 (25.33)	77 589.1 (10.13)	77 998.5 (1.28)	74 564.1 (0.10)	74 045.9 (0.97)	73 057.9
u_i_lohi.0	120 517.7 (1.33)	108 533.3 (7.28)	126 002.4 (25.44)	114 578.9 (11.00)	112 400.9 (1.29)	107 406.2 (0.09)	105 503.4 (0.97)	101 063.4
u_i_lolo.0	2785.6 (1.27)	2613.5 (6.15)	2677.0 (25.32)	2639.3 (10.13)	2735.7 (1.32)	2604.7 (0.10)	2556.0 (1.02)	2529.0
u_s_hihi.0	5 160 342.8 (1.27)	4 627 988.8 (6.07)	5 068 011.5 (25.29)	5 121 953.6 (9.00)	4 394 021.6 (1.30)	5 265 894.8 (0.32)	4 233 055.3 (1.16)	4 063 563.7
u_s_hilo.0	104 375.2 (1.58)	100 128.4 (6.23)	101 739.6 (25.62)	102 499.9 (9.20)	100 813.8 (1.26)	98 628.4 (0.42)	98 128.9 (0.74)	95 419.0
u_s_lohi.0	140 284.5 (1.25)	133 039.3 (6.05)	143 491.2 (26.21)	150 297.1 (9.90)	134 568.5 (1.32)	139 058.2 (0.40)	130 740.0 (0.73)	120 452.3
u_s_lolo.0	3806.8 (1.80)	3555.2 (6.75)	3679.6 (26.50)	3846.5 (9.28)	3695.8 (1.26)	3585.8 (0.41)	3544.4 (1.12)	3414.8
Avg. time (ms)	1.39	6.31	25.63	9.12	1.29	0.57	1.13	

Table 2
Results achieved by selected heuristics on (1024 × 32) datasets.

Dataset	Min–Min (s)	Min–Max (s)	RC (s)	Sufferage (s)	LSufferage (s)	PB (s)	TPB (s)	LP bound
A.u_c_hihi	22 508 062.4 (0.0072)	22 925 218.3 (0.0374)	26 981 108.3 (0.1738)	30 004 641.8 (0.0471)	21 398 040.3 (0.0070)	22 618 822.0 (0.0072)	20 349 518.7 (0.0143)	19 449 230.0
A.u_c_hilo	2 255 966.0 (0.0068)	2 367 409.4 (0.0375)	2 667 845.0 (0.1691)	2 816 620.5 (0.0450)	2 160 266.4 (0.0058)	2 282 607.8 (0.0047)	2 030 033.1 (0.0069)	1 951 345.1
A.u_c_lohi	2155.0 (0.0061)	2236.8 (0.0372)	2523.2 (0.1696)	2751.9 (0.0458)	2070.3 (0.0060)	1989.7 (0.0050)	1962.3 (0.0069)	1866.4
A.u_c_lolo	225.9 (0.0054)	234.9 (0.0361)	269.5 (0.1698)	288.2 (0.0453)	218.2 (0.0058)	215.5 (0.0049)	207.1 (0.0068)	198.9
A.u_i_hihi	6 367 767.6 (0.0058)	5 539 988.4 (0.0414)	6 527 383.1 (0.1801)	5 601 367.0 (0.0651)	5 907 910.4 (0.0056)	5 524 262.1 (0.0002)	5 355 933.0 (0.0037)	5 012 207
A.u_i_hilo	641 438.4 (0.0057)	502 462.8 (0.0444)	633 201.9 (0.1766)	533 545.2 (0.0673)	576 607.0 (0.0054)	538 232.0 (0.0002)	510 265.4 (0.0038)	474 404.6
A.u_i_lohi	664.7 (0.0054)	554.5 (0.0448)	615.5 (0.1761)	551.7 (0.0662)	562.7 (0.0053)	554.0 (0.0002)	541.2 (0.0049)	503.4
A.u_i_lolo	63.7 (0.0054)	54.8 (0.0413)	58.5 (0.1721)	54.8 (0.0629)	53.7 (0.0053)	53.9 (0.0002)	52.5 (0.0038)	49
A.u_s_hihi	14 125 881.6 (0.0054)	14 744 164.4 (0.0326)	15 672 428.1 (0.1675)	16 939 635.8 (0.0528)	13 874 302.2 (0.0053)	13 675 455.3 (0.0015)	12 816 629.1 (0.0041)	11 553 632
A.u_s_hilo	1 319 050.6 (0.0058)	1 391 542.1 (0.0343)	1 506 525.9 (0.1679)	1 603 159.0 (0.0501)	1 373 949.1 (0.0053)	1 286 809.0 (0.0014)	1 250 464.1 (0.0043)	1 126 556.2
A.u_s_lohi	1380.5 (0.0054)	1445.1 (0.0326)	1515.9 (0.1678)	1681.4 (0.0518)	1359.2 (0.0054)	1342.5 (0.0013)	1238.8 (0.0043)	1122.2
A.u_s_lolo	140.6 (0.0054)	148.3 (0.0331)	154.7 (0.1659)	170.2 (0.0512)	140.5 (0.0054)	132.5 (0.0014)	129.4 (0.0039)	116.7
B.u_c_hihi	6 708 228.5 (0.0055)	7 196 153.8 (0.0375)	7 755 840.0 (0.1700)	8 514 664.6 (0.0451)	6 645 054.9 (0.0053)	6 401 701.1 (0.0049)	6 273 091.6 (0.0066)	5 980 871.9
B.u_c_hilo	66 684.5 (0.0055)	70 842.6 (0.0371)	78 064.0 (0.1693)	84 876.7 (0.0451)	66 143.1 (0.0052)	62 553.1 (0.0051)	61 480.0 (0.0067)	58 942.5
B.u_c_lohi	232 011.8 (0.0055)	247 657.7 (0.0375)	275 938.9 (0.1695)	296 032.8 (0.0453)	227 227.9 (0.0053)	227 051.6 (0.0051)	216 611.4 (0.0071)	207 892.8
B.u_c_lolo	2386.3 (0.0054)	2515.8 (0.0366)	2854.3 (0.1685)	3109.5 (0.0458)	2286.2 (0.0053)	2246.9 (0.0050)	2186.0 (0.0069)	2078
B.u_i_hihi	2 164 576.7 (0.0055)	1 862 257.9 (0.0429)	2 184 682.7 (0.1779)	1 847 652.5 (0.0643)	1 885 819.5 (0.0054)	1 863 762.5 (0.0002)	1 676 943.4 (0.0037)	1 567 178.7
B.u_i_hilo	17 083.1 (0.0055)	16 211.2 (0.0401)	16 400.4 (0.1771)	16 366.2 (0.0671)	16 322.1 (0.0059)	16 647.0 (0.0002)	15 554.2 (0.0041)	14 582.3
B.u_i_lohi	56 601.2 (0.0056)	56 214.5 (0.0381)	57 057.3 (0.1755)	55 083.2 (0.0643)	56 040.5 (0.0055)	54 551.5 (0.0002)	51 250.8 (0.0044)	47 606.9
B.u_i_lolo	585.0 (0.0054)	559.1 (0.0410)	541.8 (0.1752)	535.2 (0.0661)	579.0 (0.0053)	543.5 (0.0002)	514.4 (0.0042)	477.4
B.u_s_hihi	3 967 265.9 (0.0055)	4 029 115.2 (0.0334)	4 311 309.6 (0.1694)	4 714 483.4 (0.0532)	3 880 620.3 (0.0054)	3 693 492.2 (0.0014)	3 627 964.5 (0.0035)	3 178 482.2
B.u_s_hilo	40 691.6 (0.0056)	43 454.2 (0.0327)	47 298.2 (0.1688)	50 884.3 (0.0504)	40 964.1 (0.0054)	39 322.7 (0.0014)	38 276.5 (0.0037)	33 948.7
B.u_s_lohi	135 624.6 (0.0056)	136 314.6 (0.0339)	149 379.5 (0.1664)	152 928.2 (0.0532)	128 850.1 (0.0055)	126 185.7 (0.0013)	122 973.8 (0.0035)	108 330.1
B.u_s_lolo	1333.2 (0.0055)	1439.1 (0.0347)	1540.7 (0.1681)	1686.8 (0.0521)	1432.7 (0.0054)	1279.2 (0.0020)	1255.5 (0.0042)	1128.1
Avg.time (s)	0.0057	0.0374	0.1713	0.0543	0.0055	0.0023	0.1263	

Table 3
Results achieved by selected heuristics on (2048 × 64) datasets.

Dataset	Min–Min (s)	Min–Max (s)	RC (s)	Sufferage (s)	LSufferage (s)	PB (s)	TPB (s)	LP bound
A.u_c_hihi	19 552 221.8 (0.026)	21 262 097.4 (0.249)	24 106 961.0 (1.280)	25 579 857.8 (0.297)	19 088 859.4 (0.026)	20 098 844.0 (0.0248)	17 820 097.5 (0.038)	17 141 977.4
A.u_c_hilo	1 873 134.2 (0.026)	2 027 221.2 (0.240)	2 305 994.2 (1.271)	2 478 699.3 (0.297)	1 839 290.3 (0.024)	1 783 578.9 (0.0245)	1 740 446.2 (0.032)	1 664 592.8
A.u_c_lohi	1924.7 (0.025)	2136.8 (0.239)	2395.6 (1.278)	2548.9 (0.307)	1892.9 (0.025)	1879.5 (0.0231)	1785.3 (0.033)	1695.3
A.u_c_lolo	191.6 (0.025)	207.8 (0.237)	236.6 (1.217)	254.3 (0.289)	186.1 (0.024)	183.1 (0.0224)	174.9 (0.032)	168.3
A.u_i_hihi	3 248 935.4 (0.025)	2 619 872.4 (0.256)	3 359 140.1 (1.289)	3 218 272.5 (0.417)	3 078 225.4 (0.024)	2 846 358.5 (0.0005)	2 667 259.2 (0.018)	2 366 682.1
A.u_i_hilo	365 828.6 (0.025)	295 287.7 (0.266)	361 440.5 (1.274)	315 267.5 (0.416)	323 116.6 (0.024)	321 792.3 (0.0005)	287 055.3 (0.019)	260 904.5
A.u_i_lohi	320.9 (0.024)	313.9 (0.257)	312.3 (1.264)	313.3 (0.418)	317.6 (0.024)	307.4 (0.0005)	276.1 (0.018)	255.2
A.u_i_lolo	32.3 (0.024)	28.1 (0.254)	31.8 (1.235)	32.0 (0.408)	31.0 (0.024)	31.4 (0.0005)	28.6 (0.016)	25.1
A.u_s_hihi	11 245 679.6 (0.025)	12 016 228.5 (0.228)	12 909 673.3 (1.247)	13 890 956.6 (0.369)	11 491 660.8 (0.024)	10 570 583.2 (0.0065)	10 071 471.6 (0.019)	9 050 260.8
A.u_s_hilo	1 042 948.5 (0.025)	1 097 463.5 (0.224)	1 204 146.4 (1.253)	1 307 394.5 (0.339)	1 053 603.2 (0.024)	966 817.9 (0.0096)	936 387.0 (0.023)	851 399.9
A.u_s_lohi	1056.0 (0.025)	1121.3 (0.219)	1262.5 (1.251)	1362.9 (0.357)	1088.9 (0.024)	1014.6 (0.0066)	988.2 (0.020)	888.9
A.u_s_lolo	115.3 (0.024)	119.7 (0.209)	131.0 (1.197)	138.0 (0.343)	113.1 (0.024)	109.5 (0.0063)	102.3 (0.019)	92.3
B.u_c_hihi	5 564 664.3 (0.025)	6 142 490.9 (0.242)	6 989 304.8 (1.275)	7 560 319.1 (0.296)	5 553 517.3 (0.024)	5 475 916.4 (0.0237)	5 186 179.2 (0.033)	4 975 778.8
B.u_c_hilo	59 352.8 (0.025)	64 439.2 (0.246)	74 905.3 (1.273)	79 229.7 (0.301)	58 199.8 (0.025)	58 156.9 (0.0237)	54 331.7 (0.035)	52 240.6
B.u_c_lohi	190 842.4 (0.025)	206 397.6 (0.243)	244 632.0 (1.273)	254 098.4 (0.302)	186 203.5 (0.024)	182 007.3 (0.0237)	174 844.3 (0.035)	167 381.1
B.u_c_lolo	1927.7 (0.025)	2117.2 (0.246)	2382.2 (1.281)	2559.6 (0.301)	1905.5 (0.024)	1930.4 (0.0229)	1787.3 (0.034)	1715.0
B.u_i_hihi	929 295.8 (0.025)	811 805.5 (0.266)	883 495.6 (1.278)	879 421.4 (0.415)	922 877.9 (0.024)	906 682.1 (0.0005)	832 197.8 (0.021)	735 101.5
B.u_i_hilo	10 318.4 (0.025)	9005.7 (0.262)	11 498.4 (1.281)	9047.7 (0.424)	10 089.9 (0.024)	8847.5 (0.0006)	8318.0 (0.019)	7536.3
B.u_i_lohi	34 071.0 (0.025)	28 477.7 (0.269)	32 896.9 (1.284)	32 073.6 (0.417)	32 774.1 (0.025)	30 406.8 (0.0006)	28 689.3 (0.021)	25 681.2
B.u_i_lolo	355.7 (0.025)	281.0 (0.264)	326.6 (1.267)	299.3 (0.410)	319.9 (0.024)	304.8 (0.0005)	284.2 (0.015)	250.5
B.u_s_hihi	3 293 157.1 (0.025)	3 504 763.5 (0.225)	3 839 465.7 (1.231)	4 121 618.5 (0.332)	3 332 695.8 (0.024)	3 128 983.6 (0.0065)	2 984 664.8 (0.018)	2 710 023.7
B.u_s_hilo	33 445.4 (0.024)	35 686.1 (0.223)	39 348.7 (1.244)	41 556.2 (0.349)	33 015.2 (0.025)	32 017.7 (0.0065)	30 307.6 (0.020)	27 268.0
B.u_s_lohi	111 237.4 (0.025)	117 915.2 (0.224)	131 622.9 (1.242)	142 534.7 (0.358)	110 918.2 (0.024)	106 055.4 (0.0062)	102 637.4 (0.017)	90 727.3
B.u_s_lolo	1163.8 (0.024)	1220.0 (0.224)	1333.9 (1.232)	1440.8 (0.343)	1128.9 (0.024)	1099.2 (0.0062)	1051.9 (0.017)	939.0
Avg.time (s)	0.025	0.242	1.259	0.354	0.024	0.010	0.570	

Table 4
Results achieved by selected heuristics on (4096 × 128) datasets.

Dataset	Min–Min (s)	Min–Max (s)	RC (s)	Sufferage (s)	LSufferage (s)	PB (s)	TPB (s)	LP bound
A.u_c_hihi	16 711 134.3 (0.124)	18 985 749.7 (1.887)	21 789 259.7 (10.278)	23 173 822.0 (2.028)	16 443 048.5 (0.115)	17 562 551.5 (0.112)	15 508 810.0 (0.176)	14 829 360.6
A.u_c_hilo	1 649 763.7 (0.111)	1 862 954.5 (1.641)	2 161 312.6 (9.890)	2 241 888.2 (2.010)	1 645 134.9 (0.110)	1 658 172.9 (0.110)	1 539 701.3 (0.160)	1 478 358.1
A.u_c_lohi	1631.4 (0.109)	1858.0 (1.629)	2132.2 (9.816)	2241.4 (1.975)	1631.5 (0.111)	1873.4 (0.105)	1518.2 (0.159)	1452.5
A.u_c_lolo	164.2 (0.109)	188.4 (1.590)	211.6 (9.370)	221.9 (1.868)	164.7 (0.109)	172.4 (0.106)	153.9 (0.160)	147.4
A.u_i_hihi	1 666 126.4 (0.113)	1 396 810.4 (1.669)	1 489 746.8 (10.013)	1 575 787.7 (2.594)	1 429 945.0 (0.114)	1 619 078.1 (0.002)	1 391 594.8 (0.093)	1 231 099
A.u_i_hilo	177 692.2 (0.114)	146 570.6 (1.641)	161 706.0 (10.054)	154 506.9 (2.535)	162 703.6 (0.110)	183 020.7 (0.002)	151 189.7 (0.091)	128 539.5
A.u_i_lohi	188.0 (0.111)	141.1 (1.641)	164.0 (9.858)	157.3 (2.463)	164.2 (0.108)	172.6 (0.002)	148.9 (0.102)	127.6
A.u_i_lolo	19.4 (0.106)	14.5 (1.562)	18.0 (9.241)	16.2 (2.354)	16.6 (0.107)	16.7 (0.002)	15.1 (0.071)	12.9
A.u_s_hihi	8 949 853.0 (0.119)	10 045 318.2 (1.483)	11 014 978.9 (9.870)	11 756 833.9 (2.255)	9 086 496.0 (0.109)	8 735 400.2 (0.037)	8 288 495.8 (0.087)	7 553 763.4
A.u_s_hilo	930 564.3 (0.111)	1 027 469.0 (1.508)	1 138 748.7 (9.909)	1 215 532.6 (2.297)	925 327.8 (0.109)	879 261.0 (0.031)	855 287.9 (0.093)	768 703.1
A.u_s_lohi	927.9 (0.109)	992.5 (1.442)	1119.8 (9.804)	1207.4 (2.313)	916.2 (0.111)	875.7 (0.032)	830.7 (0.090)	748.5
A.u_s_lolo	94.9 (0.108)	102.3 (1.416)	114.0 (9.306)	121.6 (2.141)	94.3 (0.108)	89.4 (0.043)	86.7 (0.076)	78.4
B.u_c_hihi	5 059 570.6 (0.110)	5 796 839.1 (1.625)	6 598 386.8 (10.043)	6 942 720.4 (2.023)	5 056 693.1 (0.110)	5 654 522.6 (0.106)	4 719 662.7 (0.164)	4 514 305.9
B.u_c_hilo	49 301.2 (0.110)	56 365.5 (1.649)	63 732.1 (9.913)	66 463.4 (2.010)	49 233.8 (0.108)	55 757.8 (0.107)	46 034.2 (0.160)	44 027
B.u_c_lohi	169 495.3 (0.110)	190 384.4 (1.688)	219 709.7 (10.214)	226 914.0 (2.011)	167 265.2 (0.110)	170 018.5 (0.111)	157 093.4 (0.163)	150 530
B.u_c_lolo	1662.3 (0.109)	1866.5 (1.662)	2148.1 (9.853)	2258.6 (1.997)	1648.8 (0.109)	1653.7 (0.109)	1543.1 (0.159)	1474
B.u_i_hihi	524 174.1 (0.111)	461 682.3 (1.624)	503 344.2 (10.021)	472 071.9 (2.524)	443 735.9 (0.110)	514 632.7 (0.001)	434 982.3 (0.099)	374 988.9
B.u_i_hilo	5381.1 (0.108)	4506.3 (1.643)	5380.7 (9.977)	4964.7 (2.508)	5023.7 (0.109)	5206.0 (0.002)	4538.3 (0.091)	3942.5
B.u_i_lohi	18 772.1 (0.113)	15 461.6 (1.642)	17 628.1 (9.922)	15 873.6 (2.489)	16 557.0 (0.109)	18 075.9 (0.002)	15 082.8 (0.103)	12 825.3
B.u_i_lolo	183.9 (0.108)	144.2 (1.582)	187.5 (9.843)	157.1 (2.473)	179.5 (0.110)	171.2 (0.002)	154.4 (0.079)	128.8
B.u_s_hihi	2 843 119.0 (0.118)	3 031 942.2 (1.484)	3 397 206.7 (9.829)	3 551 047.0 (2.267)	2 828 954.7 (0.107)	2 806 158.2 (0.031)	2 600 944.2 (0.084)	2 353 555.3
B.u_s_hilo	27 793.4 (0.109)	30 872.4 (1.483)	34 575.3 (9.730)	36 605.5 (2.235)	28 327.1 (0.108)	27 638.1 (0.031)	26 083.7 (0.086)	23 417.3
B.u_s_lohi	91 523.0 (0.108)	98 791.7 (1.448)	111 360.0 (9.851)	115 431.7 (2.284)	91 600.1 (0.109)	87 756.4 (0.031)	83 795.5 (0.090)	75 488.9
B.u_s_lolo	921.8 (0.109)	1024.9 (1.453)	1143.4 (9.754)	1194.1 (2.242)	935.6 (0.110)	906.5 (0.031)	854.0 (0.088)	771.8
Avg.time (s)	0.111	1.587	9.848	2.246	0.110	0.048	0.114	

Table 5
Results achieved by selected heuristics on (8192 × 256) datasets.

Dataset	Min–Min (s)	Min–Max (s)	RC (s)	Sufferage (s)	LSufferage (s)	PB (s)	TPB (s)	LP bound
A.u_c_hihi	14 798 375.7 (0.57)	17 346 736.3 (13.94)	19 649 354.5 (80.22)	20 086 598.9 (16.28)	14 826 007.0 (0.56)	16 315 155.0 (0.775)	13 935 511.8 (1.04)	13 338 612.5
A.u_c_hilo	1 500 182.1 (0.53)	1 753 920.4 (13.76)	2 013 425.7 (80.79)	2 058 652.4 (16.94)	1 508 866.5 (0.50)	1 709 890.9 (0.565)	1 412 881.7 (0.83)	1 352 062.2
A.u_c_lohi	1458.8 (0.50)	1715.0 (14.59)	1940.0 (80.11)	2029.7 (18.17)	1466.3 (0.55)	1559.6 (0.628)	1368.7 (0.86)	1307.6
A.u_c_lolo	149.0 (0.57)	172.8 (14.01)	197.5 (74.84)	203.2 (14.94)	148.1 (0.46)	164.5 (0.485)	138.8 (0.72)	132.6
A.u_i_hihi	878 829.5 (0.52)	781 824.9 (13.39)	830 821.3 (80.51)	788 940.8 (18.77)	811 387.6 (0.49)	997 038.0 (0.005)	768 209.2 (0.36)	634 712.8
A.u_i_hilo	85 076.7 (0.56)	72 165.3 (13.75)	80 914.7 (77.89)	77 317.0 (19.08)	77 762.7 (0.50)	99 487.6 (0.005)	77 974.6 (0.40)	63 530.6
A.u_i_lohi	96.1 (0.65)	79.0 (14.61)	95.3 (78.77)	95.2 (19.18)	88.9 (0.48)	104.6 (0.005)	77.6 (0.40)	63.7
A.u_i_lolo	8.8 (0.48)	7.5 (12.68)	8.7 (69.79)	8.4 (17.05)	8.2 (0.49)	10.7 (0.005)	7.8 (0.33)	6.4
A.u_s_hihi	8 151 521.7 (0.53)	9 185 400.5 (13.82)	10 318 419.0 (85.90)	10 828 658.9 (20.45)	8 232 565.1 (0.57)	8 231 763.5 (0.191)	7 552 141.1 (0.47)	6 812 019.5
A.u_s_hilo	787 507.8 (0.57)	873 076.4 (15.81)	985 862.9 (77.80)	1 043 244.7 (17.67)	798 414.3 (0.48)	782 963.7 (0.142)	727 980.1 (0.40)	657 203
A.u_s_lohi	796.8 (0.46)	877.1 (12.85)	1002.2 (73.93)	1071.2 (16.64)	800.4 (0.45)	832.3 (0.136)	731.7 (0.39)	661.9
A.u_s_lolo	81.3 (0.46)	90.2 (11.68)	102.0 (66.69)	106.0 (15.19)	81.7 (0.44)	84.8 (0.122)	75.0 (0.34)	67.9
B.u_c_hihi	4 460 897.9 (0.49)	5 244 866.8 (13.85)	5 974 319.9 (76.38)	6 215 689.7 (16.00)	4 473 259.0 (0.48)	5 054 532.8 (0.518)	4 199 121.3 (0.76)	4 013 215.4
B.u_c_hilo	43 670.3 (0.48)	51 609.1 (13.99)	58 956.2 (76.03)	60 943.1 (16.06)	44 032.2 (0.50)	48 220.0 (0.517)	41 150.9 (0.75)	39 256.7
B.u_c_lohi	148 102.7 (0.50)	173 202.7 (14.07)	198 694.2 (76.12)	203 270.1 (16.18)	148 176.8 (0.48)	161 097.0 (0.510)	138 922.0 (0.75)	132 570.1
B.u_c_lolo	1468.6 (0.46)	1729.5 (13.66)	1947.0 (74.85)	1981.5 (15.82)	1462.0 (0.45)	1686.1 (0.511)	1379.7 (0.76)	1319.5
B.u_i_hihi	286 800.2 (0.48)	225 075.4 (13.32)	271 117.9 (75.82)	248 651.3 (17.97)	254 240.2 (0.48)	314 229.6 (0.006)	243 134.9 (0.25)	190 045.7
B.u_i_hilo	2960.2 (0.48)	2177.5 (12.81)	2784.1 (75.47)	2406.7 (17.34)	2443.7 (0.46)	3178.5 (0.005)	2311.8 (0.28)	1894.2
B.u_i_lohi	9496.4 (0.50)	7481.8 (13.49)	9289.3 (75.75)	7887.3 (18.06)	8805.8 (0.46)	10 697.0 (0.005)	7764.9 (0.38)	6311.4
B.u_i_lolo	90.0 (0.46)	86.3 (13.37)	88.7 (74.71)	82.4 (17.88)	84.7 (0.45)	94.4 (0.005)	78.8 (0.30)	64.1
B.u_s_hihi	2 411 292.2 (0.52)	2 654 489.9 (13.13)	2 994 751.8 (75.34)	3 137 134.1 (17.15)	2 415 198.4 (0.47)	2 481 695.9 (0.139)	2 223 878.8 (0.38)	2 003 494.4
B.u_s_hilo	23 728.5 (0.47)	26 739.0 (13.26)	30 137.3 (76.12)	31 987.8 (17.67)	24 799.2 (0.49)	24 538.1 (0.151)	22 426.1 (0.46)	20 179.1
B.u_s_lohi	79 291.5 (0.55)	88 968.2 (13.74)	100 364.3 (79.09)	106 214.3 (17.76)	79 977.5 (0.47)	81 760.5 (0.142)	73 469.9 (0.45)	66 458.3
B.u_s_lolo	814.4 (0.46)	893.0 (12.86)	1000.4 (73.96)	1063.1 (17.12)	800.4 (0.48)	897.8 (0.131)	735.4 (0.37)	668.4
Avg.time (s)	0.51	13.60	76.54	17.31	0.48	0.24	0.52	

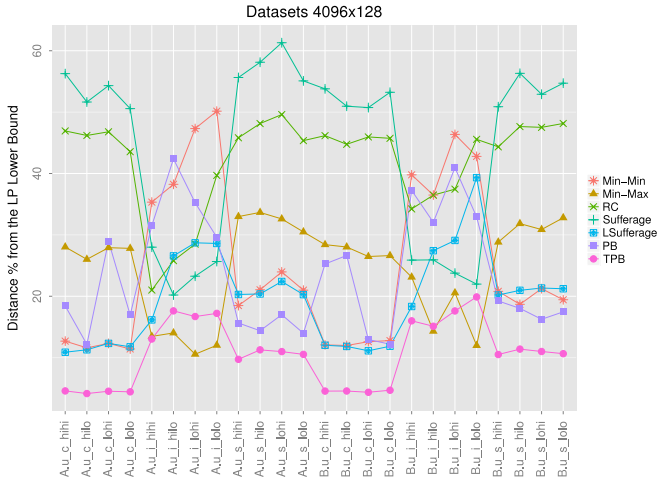

Fig. 5. LSufferage and TPB vs. other heuristics (4096 × 128).

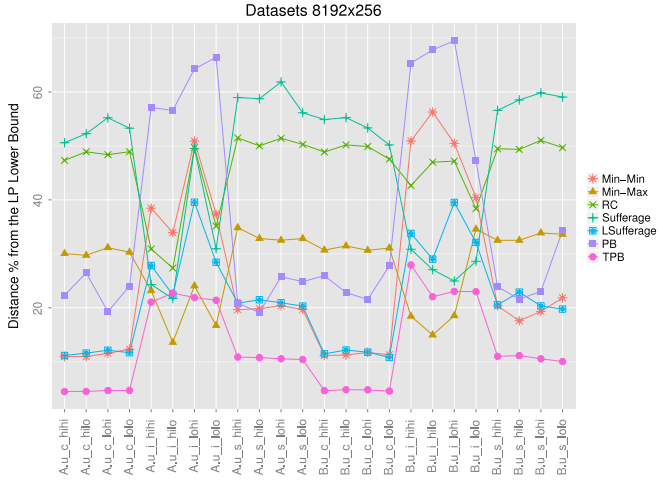

Fig. 6. LSufferage and TPB vs. other heuristics (8192 × 256).

108 instances (75%) for LSufferage, while TPB produces better results than PB in all 108 instances. Furthermore, TPB stands out as the best among the selected heuristics for most of the cases. In particular, across the 108 datasets that were used at the experiments TPB manages to achieve the best solution in 92 of them which accounts for 85.2% of all cases. Moreover, TPB finds either the best or the second best solution in all examined problem instances. Whenever TPB looses the first position this is done by Min–Max and this occurs only for some of the inconsistent type of problem instances. The other proposed heuristic, LSufferage, manages to produce competitive results, improving significantly over the original Sufferage heuristic for most of the cases. Furthermore, experiments showed in practice that TPB and LSufferage are very fast heuristics. In particular TPB and LSufferage were very close in execution times to the fastest running heuristic which was PB and noticeably faster than Min–Max, RC and Sufferage.

6.2. Experiments over Math Solver and the Column Pricing approach

A number of experiments were performed involving Math Solver and the Column Pricing approaches. As was previously mentioned, the mathematical programming solver used was Gurobi. For the case of the Column Pricing approach the Linear Programming solver of Gurobi was used running under default settings but using only a single core.

The best results achieved by the Math Solver and the Column Pricing approach can be found in https://github.com/chgogos/hcsp.

6.2.1. Experiments over small datasets

The first set of experiments involved the Braun et al. problem instances. For each problem instance Math Solver was allowed to run for 90 s. On the other hand, each run of the CPR approach was allowed to execute only for 10 s because the results obtained during this duration were already very good while provisioning extra time did not improve the results significantly. Results achieved by the CPD approach were recorded alongside with the average time needed to reach the Linear Programming lower bound of the full problem. CPR and CPD were allowed to run for 30 times over each problem instance and best, average and worst values were recorded. Math Solver, CPD and CPR were compared against state of the art approaches which were found in the bibliography. In particular, the following approaches were used for comparison: Memetic Algorithm and Tabu Search (MA+TS) [25], Ant Colony Optimization and Tabu Search (ACO+TS) [27], Tabu Search (TS) [40], parallel Cross generational Heterogeneous recombination Cataclysmic mutation(pCHC) [41], and pμ-CHC [16] which consistently reached best results among the other approaches listed. MA+TS results are reported to be obtained as the best result of 10 runs with 90 s allowed execution time per problem instance in an AMD K6 450 MHz workstation with 256 MB. ACO+TS results emerged after 3.5 h of execution time for each problem instance in a workstation with a 1.6 GHz processor. TS was executed for 10 runs for each problem instance given 100 s of execution time and the best results were recorded. pCHC and pμ-CHC results were obtained using a cluster of 4 Dell PowerEdge servers interconnected with Gigabit Ethernet. Each one of the servers was equipped with a Quad Core Xeon E5430 having 8 GB RAM and running at 2.66 GHz. For the case of pCHC the results correspond to the best values obtained after 30 runs for each problem instance while for the pμ-CHC 50 runs were executed instead. 90 s of execution time were given for each problem instance. In comparison, the processing workload of the experiments for Math Solver, CPD and CPR is either close or less to the workload demanded by the other approaches.

Table 6 summarizes the results showing that Math Solver (MS) executing for 90 s achieves the best results which are very close to the theoretical lower bounds provided by the Linear Programming relaxation. On the other hand, CPD produces better results than the best result among metaheuristics in 8 out of 12 instances, in time less than one second (0.30 s in average). Results about CPD are shown in the seventh column of Table 6 with the value inside parentheses being the time of execution needed on the test machine. Furthermore, the CPR approach running for 10 s performs better than all metaheuristics listed in this table. This behavior will be consistently manifested in all subsequent experiments and demonstrates the effectiveness of the CPR approach. The last column of the table shows the percentages that the results obtained by the CPR approach are away from the lower bounds that were found by the MS during the branch and bound operations that it performed. These bounds (column MS Bound) are tighter than the bounds found by the Linear Programming solver alone (column LP bound in Table 1), since the second one represents the calculated value at the root of the branch and bound tree. On aggregate, the average distance of CPR for all problem instances of the table is 0.32% to the LP bounds and 0.18% to the more precise IP bounds. This is another indication of the very good performance of the method and suggests that for the experiments that will be presented next the real gap values should be smaller than those included in the corresponding tables since they are computed against the LP bounds.

Table 6
Datasets Braun et al. 512 tasks, 16 processors.

Instances	MA+TS	ACO+TS	TS	pCHC	pμ-CHC	CPD best-avg-worst (s)	CPR best-avg-worst (10 s)	MS (90 s)	MS bound	CPR gap
u_c_hihi.0	7 530 020.2	7 497 200.9	7 448 640.5	7 461 819.1	7 381 570.0	7 384 514.0-7 397 471.5-7 422 994.5 (0.55)	7 360 953.7-7 368 307.8-7 376 573.5	7 358 942.8	7 350 080.2	0.15%
u_c_hilo.0	153 917.2	154 234.6	153 263.3	153 791.9	153 105.4	152 962.5-153 084.1-153 177.8 (0.36)	152 833.7-152 876.4-152 920.0	152 812.2	152 719.9	0.07%
u_c_lohi.0	245 288.9	244 097.3	241 672.7	241 513.2	239 260.0	239 479.0-239 922.0-240 068.8 (0.29)	238 762.0-238 966.2-239 226.9	238 649.0	238 304.0	0.19%
u_c_lolo.0	5173.7	5178.4	5155.0	5177.5	5147.9	5143.2-5150.0-5155.9 (0.27)	5138.0-5139.5-5141.9	5135.7	5133.6	0.08%
u_i_hihi.0	3 058 474.9	2 947 754.1	2 957 854.1	2 952 493.2	2 938 380.8	2 938 870.4-2 950 934.1-2 958 202.3 (0.27)	2 926 237.4-2 931 506.1-2 937 968.0	2 924 720.0	2 922 044.1	0.14%
u_i_hilo.0	75 108.5	73 776.2	73 692.9	73 639.8	73 387.0	73 260.6-73 362.6-73 391.4 (0.22)	73 204.5-73 244.2-73 260.6	73 202.1	73 097.3	0.15%
u_i_lohi.0	105 808.6	102 445.8	103 865.7	102 123.1	102 050.6	101 908.1-102 272.9-102 470.0 (0.22)	101 698.9-101 800.7-101 937.9	101 527.0	101 404.0	0.29%
u_i_lolo.0	2596.6	2553.5	2552.1	2548.9	2541.4	2539.7-2541.5-2542.3 (0.23)	2535.0-2536.7-2538.9	2533.4	2530.8	0.17%
u_s_hihi.0	4 321 015.4	4 162 547.9	4 168 795.9	4 198 799.5	4 103 500.3	4 108 601.9-4 131 663.2-4 200 746.8 (0.27)	4 087 583.7-4 092 676.6-4 102 477.3	4 080 153.5	4 073 743.5	0.34%
u_s_hilo.0	97 177.3	96 762.0	96 180.9	96 623.3	95 787.4	95 771.4-95 877.5-96 034.0 (0.26)	95 602.4-95 642.7-95 689.3	95 553.3	95 462.5	0.15%
u_s_lohi.0	127 633.0	123 922.0	123 407.4	123 236.9	122 083.3	121 708.6-122 239.6-122 325.4 (0.29)	121 117.4-121 339.0-121 643.0	121 000.9	120 843.3	0.23%
u_s_lolo.0	3484.1	3455.2	3450.5	3450.1	3433.5	3423.5-3429.0-3430.6 (0.38)	3421.4-3423.8-3425.8	3419.4	3416.4	0.15%

6.2.2. Experiments over larger datasets

The second set of experiments were performed using the 96 larger than Braun et al. (512 × 16) sets of problem instances introduced by Nesmachnow et al. [16]. Results are presented in Tables 7–10. Math Solver is not included in these tables since problems of such sizes proved to be very large for attacking them directly using an IP solver. CPD manages to produce competitive results that for the majority of problem instances are better than pμ-CHC which is the best performing approach that we were able to find in the bibliography for these problem instances. In particular, best values obtained from CPD outperform pμ-CHC in 104 out of 108 instances. As about the worst values obtained from CPD they are better than pμ-CHC in 81 of the 108 instances. Results for CPD are presented in the third columns of the tables and as before the value inside the parentheses is the time that CPD needed in order to reach the LP lower bound. CPD and CPR run 30 times for each problem instance and best, average and worst results were recorded. In particular, CPR was allowed to run for 20, 40, 60 and 90 s for datasets 1024 × 32, 2048 × 64, 4096 × 128 and 8192 × 256 respectively. Experiments showed that allowing for each dataset category more execution time than the above mentioned the improvements achieved over solution quality were insignificant. In all problem instances across all datasets the worst solutions achieved by CPR are better than the best solutions obtained by pμ-CHC. The results clearly put CPR in a leading position as an extremely efficient approach for solving HCSP problem instances of all sizes examined.

A pattern that can be observed on the results achieved by CPR is that the method seems to perform best for the consistent type of problems, reaching values that are very close to the lower bounds. For problems of the inconsistent and the semi-consistent type of problems the gaps are larger on average. This indicates that the latter types of problems might be considered harder to solve by the proposed algorithm than the consistent ones.

6.2.3. Temporal behavior of CPR

As it was described in Section 5, CPR is essential CPD followed by a series of solution shakings and reactivations of the Column Pricing method. Normally, the time needed to reach the lower bound by the initial call to CPD will be greater than the time needed for subsequent descents. Since the CPD has stochastic components different solutions are expected to be produced in each descent and the one that has the best value will be the final solution. This behavior is presented in Fig. 7 for the problem instance u_s_lolo.0 of dataset category Braun et al. (512 × 16). The figure shows the LP solution and the Integer solution from iteration 301 to 400. In this experiment, the allowed time of execution was 10 s and CPR performed 610 iterations in total. The figure shows that several bumps of the LP solution occur but the value of it quickly returns to high quality values close to the lower bound. The Integer solution moves in sync with the LP solution. The spikes at the graph denote the shaking of the solution that CPR performs in order to reboot the procedure. It was chosen to display only a subset of 100 iterations for the sake of better appearance. CPR might produce rather bad LP solutions that after rounding and improving could produce integer solutions with better values especially during the early iterations of each descend.

It should be noted that the limit of 90 s of execution time that is used in some of our experiments dates from the mid 2000s' and since much more powerful computers exist now the execution time required for the search can be significantly reduced.

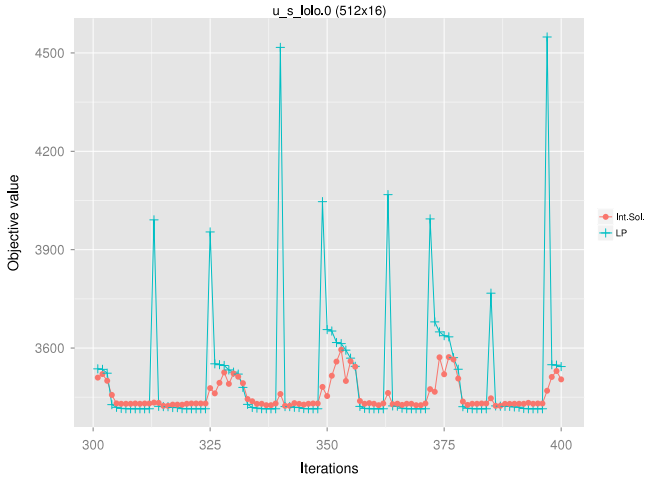

Fig. 7. Iteration 301–400 of 610 iterations.

6.3. Distances to lower bounds

In order to further demonstrate the efficiency of the methods described in this manuscript: LSufferage, TPB, MS, CPD and CPR, Table 11 was constructed. This table shows the average gaps from the known lower bounds for all problem instances of each dataset group as computed by the experiments of the above paragraphs. The new heuristics LSuferage and TPB have gaps that are closer to the bounds when compared to the heuristics Sufferage and PB respectively. On the other hand, CPD and especially CPR reach values that are very close to the lower bounds and are significantly better than the best known metaheuristic which is pμ-CHC. MS which stands for Math Solver was able to produce values only for the smallest of the datasets and expectedly reached the best results for this dataset.

7. Conclusions

In this paper, the problem of scheduling independent tasks to heterogeneous computers (HCSP) was examined. This problem has received attention by researchers in the recent years due to the trend that exists where several computers/processors/cores are becoming part of a greater entity capable of servicing a great number of tasks. HCSP might be considered as an abstraction of the problem faced by several installations, since for each task and processor combination only a single value represents the workload, tasks are not preemptive, dependencies between tasks cannot be defined and no priorities among tasks exist. Nevertheless, the problem by itself is important since it can be used as a test bed for examining the behavior of several algorithmic approaches and certain systems might consider the problem definition of HCSP suitable for their job requirements.

Two main contributions were made in this paper. First, two new heuristics, LSufferage and TPB, were developed that exhibit excellent behavior across an extended range of problem instances. LSufferage is a fast heuristic that makes use of lists of processors sorted by execution time for each task. It improves the Sufferage algorithm not only on execution time but on the quality of the produced solutions as well. TPB is the other new proposed heuristic which outperforms by a great margin other state of the art heuristics including the PB heuristic that TPB is inspired from. The second contribution is CPR which is a hybrid mathematical Column Pricing and heuristic approach that progresses by decomposing the original problem to smaller ones. A mathematical model of each subproblem is formulated and a LP solver is used in order to solve it. Then, the reduced cost values of non basic variables are examined

Table 7
Datasets 1024 tasks, 32 processors.

Instance	pμ-CHC	CPD best-avg-worst (s)	CPR best-avg-worst (20 s)	LP bound	CPR gap
A.u_c_hihi	19 676 858.3	19 582 333.9-19 624 513.9-19 647 917.5 (1.40)	19 517 000.9-19 538 501.5-19 579 665.1	19 449 230.0	0.35%
A.u_c_hilo	1 969 980.0	1 961 198.5-1 965 550.7-1 966 054.4 (1.06)	1 958 623.5-1 960 746.6-1 962 378.5	1 951 345.1	0.37%
A.u_c_lohi	1887.3	1878.0-1883.2-1888.5 (1.32)	1872.6-1874.8-1876.6	1866.4	0.33%
A.u_c_lolo	*201.2*	200.0-200.7-201.2 (1.37)	199.6-199.8-200.0	198.9	0.35%
A.u_i_hihi	5 126 273.0	5 065 657.0-5 082 144.2-5 109 083.1 (0.90)	5 052 797.0-5 058 170.7-5 063 608.4	5 012 207.0	0.81%
A.u_i_hilo	485 189.8	479 285.6-479 676.4-479 747.1 (1.01)	477 555.6-478 373.5-478 956.7	474 404.6	0.66%
A.u_i_lohi	513.8	509.9-511.3-512.8 (0.82)	507.1-508.3-509.5	503.4	0.74%
A.u_i_lolo	50.2	49.5-49.7-49.9 (1.04)	49.4-49.4-49.5	49.0	0.82%
A.u_s_hihi	11 837 170.0	11 742 241.0-11 828 018.7-11 904 941.4 (1.68)	11 674 835.2-11 712 126.9-11 742 031.2	11 553 632.0	1.05%
A.u_s_hilo	1 148 940.6	1 138 655.3-1 141 685.9-1 145 675.7 (1.78)	1 135 426.1-1 137 453.8-1 138 439.2	1 126 556.2	0.79%
A.u_s_lohi	1152.5	1137.1-1140.0-1140.4 (1.62)	1130.9-1133.1-1136.4	1122.2	0.78%
A.u_s_lolo	118.9	117.8-118.0-118.1 (1.70)	117.5-117.7-117.9	116.7	0.69%
B.u_c_hihi	6 049 220.5	6 010 267.4-6 037 118.2-6 058 555.0 (1.28)	6 002 312.3-6 007 743.8-6 012 900.3	5 980 871.9	0.36%
B.u_c_hilo	59 679.5	59 310.3-59 478.0-59 584.3 (1.16)	59 127.7-59 186.8-59 267.1	58 942.5	0.31%
B.u_c_lohi	210 005.1	208 929.2-209 423.3-210 001.4 (1.06)	208 678.2-208 807.3-208 974.8	207 892.8	0.38%
B.u_c_lolo	2100.0	2089.7-2096.9-2110.1 (1.17)	2085.0-2087.7-2089.1	2078.0	0.34%
B.u_i_hihi	1 616 697.4	1 589 369.4-1 595 313.3-1 595 793.1 (0.68)	1 582 264.6-1 584 306.3-1 586 242.8	1 567 178.7	0.96%
B.u_i_hilo	14 993.2	14 746.2-14 803.5-14 839.4 (0.78)	14 698.1-14 721.2-14 745.9	14 582.3	0.79%
B.u_i_lohi	49 060.5	48 278.6-48 664.6-48 859.9 (1.00)	48 096.6-48 234.5-48 356.6	47 606.9	1.03%
B.u_i_lolo	487.5	482.7-483.8-486.0 (0.85)	481.7-482.4-482.6	477.4	0.90%
B.u_s_hihi	3 255 266.8	3 244 457.6-3 257 142.1-3 264 552.1 (1.50)	3 220 439.4-3 231 767.3-3 244 469.2	3 178 482.2	1.32%
B.u_s_hilo	34 675.2	34 387.1-34 562.8-34 776.1 (1.87)	34 260.9-34 332.6-34 410.6	33 948.7	0.92%
B.u_s_lohi	110 749.7	110 133.3-110 452.3-110 913.8 (2.08)	109 311.3-109 613.4-109 914.7	108 330.1	0.91%
B.u_s_lolo	1153.1	1140.0-1141.2-1142.5 (2.25)	1135.2-1137.8-1140.2	1128.1	0.63%

Table 8
Datasets 2048 tasks, 64 processors.

Instance	pμ-CHC	CPD best-avg-worst (s)	CPR best-avg-worst (40 s)	LP bound	CPR gap
A.u_c_hihi	17 474 552.0	17 259 517.5-17 289 689.5-17 331 342.0 (3.43)	17 224 612.6-17 237 298.0-17 248 268.8	17 141 977.4	0.48%
A.u_c_hilo	1 692 750.0	1 672 757.3-1 676 535.9-1 679 678.4 (3.88)	1 671 106.2-1 672 714.2-1 673 405.2	1 664 592.8	0.39%
A.u_c_lohi	1731.9	1713.4-1721.0-1723.1 (3.77)	1705.8-1709.1-1712.1	1695.3	0.62%
A.u_c_lolo	171.7	169.6-170.5-171.0 (3.45)	169.2-169.4-169.6	168.3	0.53%
A.u_i_hihi	2 477 753.9	2 406 303.9-2 417 333.8-2 424 534.5 (2.49)	2 399 354.7-2 402 475.6-2 407 361.0	2 366 682.1	1.38%
A.u_i_hilo	272 181.1	265 153.0-265 437.1-265 683.3 (3.02)	264 342.1-264 724.7-265 103.2	260 904.5	1.32%
A.u_i_lohi	265.6	258.7-259.0-259.4 (2.52)	257.9-258.2-258.4	255.2	1.06%
A.u_i_lolo	26.3	25.4-25.4-25.5 (2.36)	25.4-25.4-25.4	25.1	1.20%
A.u_s_hihi	9 359 727.3	9 236 272.6-9 257 847.4-9 287 308.1 (4.65)	9 176 999.7-9 208 813.7-9 235 145.9	9 050 260.8	1.40%
A.u_s_hilo	878 838.4	867 177.9-869 622.1-872 427.0 (3.15)	862 393.3-864 779.4-866 289.9	851 399.9	1.29%
A.u_s_lohi	911.8	902.4-905.7-907.1 (3.15)	899.5-900.5-901.5	888.9	1.19%
A.u_s_lolo	95.0	94.3-95.1-95.6 (4.75)	93.8-94.1-94.5	92.3	1.63%
B.u_c_hihi	5 085 005.2	5 023 382.9-5 055 806.2-5 080 707.2 (2.65)	5 004 519.1-5 014 468.5-5 023 583.7	4 975 778.8	0.58%
B.u_c_hilo	53 236.9	52 636.0-52 947.5-53 408.2 (2.98)	52 569.8-52 612.1-52 695.5	52 240.6	0.63%
B.u_c_lohi	170 659.4	168 949.5-169 718.8-170 086.8 (3.05)	168 264.4-168 649.5-168 909.9	167 381.1	0.53%
B.u_c_lolo	1749.4	1725.9-1732.3-1735.4 (3.19)	1724.3-1725.6-1727.0	1715.0	0.54%
B.u_i_hihi	763 701.5	744 840.5-746 433.0-746 958.2 (3.00)	743 337.2-744 395.7-745 046.8	735 101.5	1.12%
B.u_i_hilo	7859.0	7692.8-7714.8-7729.8 (2.41)	7650.3-7665.4-7678.4	7536.3	1.51%
B.u_i_lohi	26 769.6	26 031.9-26 122.6-26 204.5 (1.96)	25 969.3-26 001.2-26 030.1	25 681.2	1.12%
B.u_i_lolo	261.8	254.2-255.0-256.1 (2.70)	253.5-253.9-254.2	250.5	1.20%
B.u_s_hihi	2 789 531.9	2 745 624.0-2 762 878.2-2 775 426.4 (4.50)	2 738 252.4-2 746 432.7-2 750 960.0	2 710 023.7	1.04%
B.u_s_hilo	28 170.9	27 750.4-27 854.4-27 877.2 (3.07)	27 627.7-27 687.0-27 753.0	27 268.0	1.32%
B.u_s_lohi	93 798.0	92 559.1-93 183.1-93 858.0 (3.89)	92 113.4-92 303.3-92 521.3	90 727.3	1.53%
B.u_s_lolo	969.7	958.2-965.1-971.4 (3.46)	952.3-956.3-958.8	939.0	1.42%

and based on these values new columns are incorporated in the next mathematical problem formulation. Whenever a LP model is solved, the solution is heuristically transformed to an integer one and subsequently improved by applying a series of local search heuristics. Whenever the lower bound for the Linear Programming problem is reached part of the solution is randomly perturbed and the procedure repeats provided there is still available time. The best integer solution produced during the whole process is returned as the solution. CPR is able to solve in reasonable time arbitrary large problem instances and produce superior results when compared to all other methods known to us. During the experiments the execution time allowed for CPR was from 10 to 90 s, depending on problem size. CPR managed to reach best values for all problem instances. Moreover, the hardware requirements of CPR is only a fairly fast computer and not some kind of a High Performance Computing (HPC) infrastructure like the one used in the pμ-CHC approach.

Several directions exist on extending this work. The multi-objective nature of the problem could have been one of them while a second one could have been the adaptation of the CPR approach for scheduling tasks that have dependencies regarding the order of execution among them. A third direction could have been the incorporation of CPR in a metaheuristic running over a HPC infrastructure.

Acknowledgment

This work was co-funded by the European Union under the 7th Framework Program under grant agreement ICT-287 733, project "Architecture oriented parallelization for high performance embedded Multicore systems using scilAb (ALMA)".

Table 9
Datasets 4096 tasks, 128 processors.

Instance	pμ-CHC	CPD best-avg-worst (s)	CPR best-avg-worst (60 s)	LP bound	CPR gap
A.u_c_hihi	15 260 752.4	15 016 637.5-15 115 859.0-15 178 609.0 (9.84)	14 945 478.5-14 989 065.2-15 033 785.4	14 829 361	0.78%
A.u_c_hilo	1 520 225.1	1 495 420.4-1 497 560.9-1 499 865.7 (8.40)	1 489 722.0-1 491 920.6-1 492 136.6	1 478 358.1	0.77%
A.u_c_lohi	1493.8	1473.7-1482.4-1485.2 (11.41)	1462.2-1469.7-1476.3	1452.5	0.67%
A.u_c_lolo	151.1	148.6-149.3-149.6 (8.36)	148.6-148.8-148.9	147.4	0.81%
A.u_i_hihi	1 295 054.0	1 247 821.8-1 250 357.8-1 257 593.8 (9.33)	1 246 484.1-1 247 247.2-1 248 152.1	1 231 099	1.25%
A.u_i_hilo	135 985.3	130 768.5-130 939.0-131 207.2 (10.08)	130 356.7-130 544.8-130 696.2	128 539.5	1.41%
A.u_i_lohi	135.3	129.6-130.0-130.4 (10.53)	129.5-129.7-129.8	127.6	1.49%
A.u_i_lolo	13.6	13.1-13.1-13.1 (9.76)	13.1-13.1-13.1	12.9	1.55%
A.u_s_hihi	7 831 962.8	7 715 347.2-7 738 120.3-7 742 103.4 (8.55)	7 670 057.5-7 699 322.7-7 719 398.0	7 553 763.4	1.54%
A.u_s_hilo	799 499.4	788 445.2-794 214.7-800 150.2 (9.68)	784 005.7-787 250.7-790 686.7	768 703.1	1.99%
A.u_s_lohi	778.3	770.3-771.9-776.5 (8.07)	763.9-768.0-770.1	748.5	2.06%
A.u_s_lolo	81.6	80.0-80.2-80.4 (10.39)	79.7-79.9-80.1	78.4	1.66%
B.u_c_hihi	4 649 566.5	4 567 357.1-4 580 491.9-4 585 172.0 (11.97)	4 557 909.3-4 567 556.4-4 573 615.8	4 514 305.9	0.97%
B.u_c_hilo	45 142.7	44 509.6-44 650.5-44 740.0 (10.28)	44 353.4-44 497.3-44 551.0	44 027	0.74%
B.u_c_lohi	154 504.7	152 419.8-152 531.1-152 578.9 (8.87)	151 723.9-152 039.9-152 275.4	150 530	0.79%
B.u_c_lolo	1516.2	1491.0-1498.1-1501.7 (10.97)	1486.0-1491.1-1494.7	1474	0.81%
B.u_i_hihi	398 655.1	381 007.4-381 842.0-382 240.4 (10.39)	379 949.0-380 604.9-380 972.8	374 988.9	1.32%
B.u_i_hilo	4174.5	3999.4-4013.2-4019.2 (10.24)	3996.0-4002.5-4007.4	3942.5	1.36%
B.u_i_lohi	13 614.6	13 047.9-13 066.6-13 087.2 (9.84)	13 010.6-13 026.3-13 042.3	12 825.3	1.44%
B.u_i_lolo	136.3	131.1-131.4-131.5 (7.54)	130.8-131.1-131.3	128.8	1.55%
B.u_s_hihi	2 437 604.5	2 394 103.9-2 403 638.1-2 419 970.9 (10.88)	2 385 919.8-2 393 379.4-2 397 627.3	2 353 555.3	1.38%
B.u_s_hilo	24 353.7	23 975.3-24 071.0-24 199.1 (10.65)	23 846.9-23 898.6-23 944.8	23 417.3	1.83%
B.u_s_lohi	78 296.8	77 137.2-77 366.8-77 531.8 (9.44)	76 669.6-76 949.5-77 082.2	75 488.9	1.56%
B.u_s_lolo	800.7	792.8-796.8-802.7 (8.62)	787.1-789.7-792.6	771.8	1.98%

Table 10
Datasets 8192 tasks, 256 processors.

Instance	pμ-CHC	CPD best-avg-worst (s)	CPR best-avg-worst (90 s)	LP bound	CPR gap
A.u_c_hihi	13 708 686.6	13 524 643.3-13 564 712.3-13 574 827.9 (48.98)	13 423 716.7-13 536 851.0-13 565 781.8	13 338 612.5	0.64%
A.u_c_hilo	1 388 689.3	1 371 198.9-1 376 615.8-1 383 011.4 (44.52)	1 368 618.6-1 372 285.3-1 378 014.4	1 352 062.2	1.22%
A.u_c_lohi	1344.2	1327.5-1332.2-1335.5 (46.07)	1318.0-1329.6-1335.1	1307.6	0.80%
A.u_c_lolo	136.6	134.4-134.8-135.1 (37.76)	134.3-134.5-134.9	132.6	1.28%
A.u_i_hihi	690 223.8	645 719.2-647 146.6-648 148.5 (40.51)	645 368.2-646 101.0-646 780.8	634 712.8	1.68%
A.u_i_hilo	68 428.1	64 696.0-64 832.9-64 967.7 (40.80)	64 590.3-64 697.0-64 762.3	63 530.6	1.67%
A.u_i_lohi	68.9	64.8-64.9-65.1 (24.30)	64.8-64.8-64.9	63.7	1.73%
A.u_i_lolo	6.9	6.5-6.5-6.5 (49.60)	6.5-6.5-6.5	6.4	1.56%
A.u_s_hihi	7 112 313.0	7 000 318.3-7 037 453.3-7 074 562.9 (33.43)	6 956 767.0-7 003 188.4-7 029 141.6	6 812 019.5	2.12%
A.u_s_hilo	685 350.9	673 210.5-677 037.3-680 344.6 (33.79)	671 414.0-672 815.4-674 861.8	657 203.0	2.16%
A.u_s_lohi	691.5	678.6-686.4-691.6 (39.21)	674.3-678.6-682.5	661.9	1.87%
A.u_s_lolo	70.9	69.7-70.0-70.1 (28.26)	69.4-69.6-69.9	67.9	2.21%
B.u_c_hihi	4 136 265.2	4 073 908.6-4 100 614.8-4 119 154.0 (85.76)	4 063 343.9-4 068 960.2-4 076 872.3	4 013 215.4	1.25%
B.u_c_hilo	40 410.0	39 876.2-39 992.1-40 336.1 (53.05)	39 575.7-39 882.4-39 992.6	39 256.7	0.81%
B.u_c_lohi	136 499.6	134 303.1-134 621.1-134 831.1 (44.95)	134 182.4-134 419.6-134 752.7	132 570.1	1.22%
B.u_c_lolo	1357.0	1341.2-1345.3-1348.9 (46.79)	1338.0-1342.9-1348.8	1319.5	1.40%
B.u_i_hihi	205 347.6	193 612.0-193 979.2-194 248.8 (22.57)	193 171.0-193 516.9-193 788.3	190 045.7	1.64%
B.u_i_hilo	2043.0	1929.4-1933.9-1938.0 (40.98)	1925.5-1928.3-1930.7	1894.2	1.65%
B.u_i_lohi	6812.3	6439.2-6443.0-6443.7 (47.08)	6427.3-6435.9-6443.7	6311.4	1.84%
B.u_i_lolo	69.2	65.4-65.5-65.7 (52.52)	65.3-65.4-65.5	64.1	1.87%
B.u_s_hihi	2 087 688.3	2 059 603.2-2 066 718.0-2 074 121.2 (30.47)	2 045 600.2-2 055 082.2-2 063 232.7	2 003 494.4	2.10%
B.u_s_hilo	21 004.6	20 762.5-20 885.2-20 982.0 (31.35)	20 582.1-20 708.7-20 828.4	20 179.1	2.00%
B.u_s_lohi	69 347.8	68 760.7-69 200.5-69 597.1 (30.64)	68 049.7-68 386.4-68 950.2	66 458.3	2.39%
B.u_s_lolo	696.3	683.2-687.7-694.3 (29.78)	680.9-683.0-685.2	668.4	1.87%

Table 11
Gaps to lower bounds.

Datasets	Min–Min	Sufferage	LSufferage	PB	TPB	pμ-CHC	CPD	CPR	MS
512 × 16	13.83%	17.37%	8.76%	7.25%	3.33%	0.50%	0.42%	0.18%	0.11%
1024 × 32	21.00%	35.27%	16.18%	12.98%	7.90%	1.96%	1.04%	0.69%	–
2048 × 64	23.21%	42.51%	20.35%	16.11%	9.08%	3.15%	1.38%	1.04%	–
4096 × 128	24.95%	44.21%	19.77%	23.64%	10.62%	4.15%	1.70%	1.32%	–
8192 × 256	25.36%	47.20%	21.34%	36.61%	12.75%	5.07%	2.01%	1.62%	–

References

[1] I. Foster, C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure. Vol. 1, Morgan Kaufmann, 2004.

[2] J.Y. Leung, Handbook of Scheduling: Algorithms, Models, and Performance Analysis, CRC Press, 2004.

[3] M. Drozdowski, Scheduling multiprocessor tasks an overview, European J. Oper. Res. 94 (2) (1996) 215–230.

[4] H. El-Rewini, T.G. Lewis, H.H. Ali, Task Scheduling in Parallel and Distributed Systems, Prentice-Hall, Inc., 1994.

[5] Y.-K. Kwok, I. Ahmad, Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm, J. Parallel Distrib. Comput. 47 (1) (1997) 58–77.

[6] R. Buyya, D. Abramson, J. Giddy, H. Stockinger, Economic models for resource management and scheduling in grid computing, Concurr. Comput.: Pract. Exper. 14 (13–15) (2002) 1507–1542.

[7] F. Xhafa, L. Barolli, A. Durresi, Batch mode scheduling in grid systems, Int. J. Web Grid Serv. 3 (1) (2007) 19–37.

[8] F. Xhafa, J. Carretero, L. Barolli, A. Durresi, Immediate mode scheduling in grid systems, Int. J. Web Grid Serv. 3 (2) (2007) 219–236.

[9] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman & Co, New York, NY, USA, 1979.

[10] F. Xhafa, A. Abraham, Computational models and heuristic methods for grid scheduling problems, Future Gener. Comput. Syst. 26 (4) (2010) 608–621.

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

[12] R. Sahu, A.K. Chaturvedi, Many-objective comparison of twelve grid scheduling heuristics, Int. J. Comput. Appl. 13 (6) (2011) 9–17.

[13] C. Valouxis, C. Gogos, P. Alefragis, G. Goulas, N. Voros, E. Housos, DAG scheduling using integer programming in heterogeneous parallel execution environments. in: 6th Multidisciplinary International Scheduling Conference: Theory and Applications, 2013. MISTA 2013, Gent, Belgium. 2013, pp. 392–401.

[14] S. Ali, H.J. Siegel, M. Maheswaran, D. Hensgen, Task execution time modeling for heterogeneous computing systems, in: Heterogeneous Computing Workshop, 2000.(HCW 2000) Proceedings. 9th Heterogeneous Computing Workshop, IEEE, 2000, pp. 185–199.

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

[16] S. Nesmachnow, H. Cancela, E. Alba, A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling, Appl. Soft Comput. 12 (2) (2012) 626–639.

[17] M. Canabé, S. Nesmachnow, Parallel implementations of the MinMin heterogeneous computing scheduler in GPU, CLEI Electron. J. 15 (3) (2012).

[18] H.H. Hoos, T. Stützle, Stochastic Local Search: Foundations & Applications, Elsevier, 2004.

[19] S. Iturriaga, S. Nesmachnow, F. Luna, E. Alba, A parallel local search in cpu/gpu for scheduling independent tasks on large heterogeneous computing systems, J. Supercomput. 71 (2) (2015) 648–672.

[20] A.A.P. Kazem, A.M. Rahmani, H.H. Aghdam, A modified simulated annealing algorithm for static task scheduling in grid computing, in: 2008. ICCSIT'08. International Conference on Computer Science and Information Technology, IEEE, 2008, pp. 623–627.

[21] V. Di Martino, M. Mililotti, Sub optimal scheduling in a grid using genetic algorithms, Parallel Comput. 30 (5) (2004) 553–565.

[22] F.A. Omara, M.M. Arafa, Genetic algorithms for task scheduling problem, J. Parallel Distrib. Comput. 70 (1) (2010) 13–22.

[23] A.J. Page, T.M. Keane, T.J. Naughton, Multi-heuristic dynamic task allocation using genetic algorithms in a heterogeneous distributed system, J. Parallel Distrib. Comput. 70 (7) (2010) 758–766.

[24] R. Cheng, M. Gen, Parallel machine scheduling problems using memetic algorithms, Comput. Ind. Eng. 33 (3) (1997) 761–764.

[25] F. Xhafa, A Hybrid Evolutionary Heuristic for Job Scheduling on Computational Grids, in: Studies in Computational Intelligence, vol. 75, Springer, Berlin, Heidelberg, 2007, pp. 269–311. (Chapter 11).

[26] M. Pedemonte, S. Nesmachnow, H. Cancela, A survey on parallel ant colony optimization, Appl. Soft Comput. 11 (8) (2011) 5181–5197.

[27] G. Ritchie, J. Levine, A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments, in: Proceedings of the 23rd Workshop of the UK Planning and Scheduling Special Interest Group, 2004, pp. 178–183.

[28] T. Davidović, M. Šelmić, D. Teodorović, D. Ramljak, Bee colony optimization for scheduling independent tasks to identical processors, J. Heuristics 18 (4) (2012) 549–569.

[29] M.K. Rafsanjani, A.K. Bardsiri, A new heuristic approach for scheduling independent tasks on heterogeneous computing systems, Int. J. Mach. Learn. Comput. 2 (4) (2012) 371–376.

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

[31] R. Duan, R. Prodan, T. Fahringer, Performance and cost optimization for multiple large-scale grid workflow applications, in: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, 2007. SC'07, IEEE, 2007, pp. 1–12.

[32] C. Liu, S. Baskiyar, A general distributed scalable grid scheduler for independent tasks, J. Parallel Distrib. Comput. 69 (3) (2009) 307–314.

[33] E. Kartal Tabak, B. Barla Cambazoglu, C. Aykanat, Improving the performance of independenttask assignment heuristics minmin, maxmin and sufferage, IEEE Trans. Parallel Distrib. Syst. 25 (5) (2014) 1244–1256.

[34] P. Ezzatti, M. Pedemonte, A. Martin, An efficient implementation of the min-min heuristic, Comput. Oper. Res. 40 (11) (2013) 2670–2676.

[35] H. Izakian, A. Abraham, V. Snasel, Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments, in: Computational Sciences and Optimization, 2009. CSO 2009. International Joint Conference on Computational Sciences and Optimization. Vol. 1, IEEE, 2009, pp. 8–12.

[36] M.-Y. Wu, W. Shu, A high-performance mapping algorithm for heterogeneous computing systems, in: Parallel and Distributed Processing Symposium., Proceedings 15th International, IEEE, 2001, p. 6.

[37] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, R.F. Freund, Dynamic mapping of a class of independent tasks onto heterogeneous computing systems, J. Parallel Distrib. Comput. 59 (2) (1999) 107–131.

[38] A.K. Chaturvedi, R. Sahu, New heuristic for scheduling of independent tasks in computational grid, Int. J. Grid Distrib. Comput. 4 (3) (2011) 25–36.

[39] I. Gurobi Optimization, Gurobi optimizer reference manual, 2015. URL: http://www.gurobi.com.

[40] F. Xhafa, J. Carretero, E. Alba, B. Dorronsoro, Design and evaluation of tabu search method for job scheduling in distributed environments, in: 2008. IPDPS 2008. IEEE International Symposium on Parallel and Distributed Processing, IEEE, 2008, pp. 1–8.

[41] S. Nesmachnow, H. Cancela, E. Alba, Heterogeneous computing scheduling with evolutionary algorithms, Soft Comput. 15 (4) (2011) 685–701.

Christos Gogos is a faculty member of the Department of Computer and Informatics at the Technological Educational Institute of Epirus. He received his diploma in computer engineering and informatics in 1993 from the University of Patras. In 1998 he received a master's degree in High Performance Algorithms from the University of Athens. A Ph.D. was awarded to him in 2009 from the University of Patras for his thesis on the research area of solving hard combinatorial optimization problems. He has co-authored over 30 articles in refereed scientific journals & conference proceedings. He also has significant working experience in software engineering and programming.

Christos Valouxis earned his Ph.D. degree in the Department of Electrical and Computer Engineering of the University of Patras. He has published several articles in the area of optimization, scheduling and timetabling. His main research interests are algorithm engineering, combinatorial optimization, column generation and the interaction between OR and CP approaches.

Panayiotis Alefragis holds a Diploma and Ph.D. in Electrical & Computer Engineering from University of Patras. His research interests include software engineering, parallel/distributed computing, programming languages and compilers, resource scheduling algorithms, integer and combinatorial optimization and embedded design. He has co-authored over 30 articles in refereed scientific journals & conference proceedings and has extensive research and industrial experience in the above areas. He is currently faculty of the Dept. of Computer and Informatics Engineering of the TEI of Western Greece. He is a member of the IEEE and the Engineering Chamber of Greece.

George Goulas holds a diploma and Ph.D. from the Electrical and Computer Engineering Department, University of Patras. Currently, he is a postdoctoral researcher at Computer and Informatics Engineering Department, Technological Educational Institute of Western Greece and at the Electrical and Computer Engineering Department, University of Patras. His research interests include multicore and manycore systems, grid and cloud computing, web services, metaheuristics and timetabling. Dr. Goulas with Dr. Valouxis, Dr. Gogos, Dr. Alefragis and Prof. Housos submitted the software solution that won the first place on all three tracks of the 1st International Nurse Rostering Competition 2010.

Nikolaos Voros received his Diploma in Computer and Informatics Engineering, in 1996, and his Ph.D. degree, in 2001, from University of Patras, Greece. His research interests fall in the area of embedded system design and hardware-software co-design. Currently, he is Assistant Professor at the Technological Educational Institute of Western Greece, Computer & Informatics Engineering Department where he is leading the Embedded Systems and Applications Lab. During the last years, he has participated in more than 15 research projects funded by the European Commission while he has also published a significant number of refereed articles in well-known international journals and conferences.

Efthymios Housos is Professor at the Department of Electrical and Computer Engineering at the University of Patras. His research interests are in the areas of scheduling, discrete optimization, parallel & distributed processing and computer methods for the analysis and design of engineering systems. He has great experience in client focused business analysis, strategic planning and management.