Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 302 (2014) 53–72

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Hybrid GRASP Heuristics to Solve an Unrelated Parallel Machine Scheduling Problem with Earliness and Tardiness Penalties

João Paulo de C. M. Nogueira[2] José Elias C. Arroyo[1,3]
Harlem Mauricio M. Villadiego[4] Luciana B. Gonçalves [5]

Computer Science Department
Federal University of Viçosa
Viçosa - MG - Brazil

Abstract

This paper considers an unrelated parallel machine scheduling problem with the objective of minimizing the total earliness and tardiness penalties. Machine and job-sequence dependent setup times and idle times are considered. Since the studied problem is NP-Hard, we test the applicability of algorithms based on Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic to determine near-optimal solutions. We propose three different heuristics. The first is a simple GRASP heuristic, the second heuristic includes an intensification procedure based on Path Relinking technique, and the third uses an Iterated Local Search (ILS) heuristic in the local search phase of the GRASP algorithm. The results obtained by the heuristics are compared using a set of small, medium and large instances. Comprehensive computational and statistical analyses are carried out in order to compare the performance of the algorithms.

Keywords: parallel machine scheduling, earliness and tardiness penalties, combinatorial optimization, heuristic algorithms, local search, metaheuristics.

1 Introduction

Production scheduling is an important decision-making in operational level that plays a crucial role in manufacturing and services industries. Scheduling problems deal with the allocation of available resources to jobs over given time periods and the goal is to optimize one or more objectives (or criteria) [38]. These problems are extensively investigated in the literature [9]. It occurs mainly by two aspects: the

[1] This research was funded by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and the Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG).

[2] Email: jpcmnogueira@gmail.com

[3] Email: jarroyo@dpi.ufv.br

[4] Email: harlem.villadiego3@ufv.br

[5] Email: lbrugiolo@ufv.br

http://dx.doi.org/10.1016/j.entcs.2014.01.020

first one concerns their practical importance, with various applications in several industries, like chemical, metallurgic and textile industries. The second aspect is about the difficulty for solving the majority problems of this class.

Among different production scheduling problems, parallel machine scheduling (PMS) is a typical scheduling problem with extensive practical relevance. Cheng and Sin [11] present a survey of research conducted on PMS problems.

PMS problems can be defined by a set of n jobs that need to be processed by a set of m parallel machines. The objective is to schedule jobs (each job is to be assigned to exactly one of the m machines) so that one or more criterion is minimized. When the processing time of each job is the same on all those m machines, the problem is said to be identical PMS problem. In PMS problems, the most studied optimization criterion is the minimization of the maximum completion time of the schedule, a criterion that is known as makespan. Garey and Johnson [17] showed that minimizing the makespan on $m = 2$ identical machines is a NP-hard problem.

There are two other types of problems in PMS: uniform and unrelated PMS. Unrelated parallel machines can be characterized as machines that perform the same function but have different capabilities or capacities. The processing times of the jobs depend on the machine to which they are assigned to. Unrelated parallel machines is the most realistic case which is also a generalization of the uniform and identical machines cases.

Unrelated parallel machines scheduling (UPMS) problems have been much less studied in the literature [33]. Exact and approximation algorithms for makespan minimization have been proposed by Van de Velde [48] and Martello et al. [34]. Local search and heuristics methods have also been employed for makespan minimization [18], [37], [13], [30]. Recently, other performance criteria have been considered in the UPMS problem. The total weighted completion times is minimized by Lin et al. [30] and Rodriguez et al. [41], and the total weighted tardiness minimization is considered by Liaw et al. [28] and Lin et al. [30].

Some solution approaches have been proposed to solve the UPMS problem with machine and job sequence dependent setup times. In this case, the setup time is different for each pair of jobs and each machine. That is, machines has different setup times and the setup time on machine k between jobs i and j is different than setup time on the same machine between jobs j and i ($s_{i,j,k} \neq s_{j,i,k}$) . The the UPMS problem with sequence dependent setup times has been less studied and only a few papers can be found in the literature [3]. The most commonly used methods for this problem are metaheuristics. For minimizing the makespan, Franca et al. [16] suggest a tabu search algorithm, Vallada and Ruiz [47] proposed a genetic algorithm, and Changand and Chen [10] developed a metaheuristic by integrating dominance properties with genetic algorithm. In Kim et al. [24] and Kim and Shin [23] simulated annealing and tabu search algorithms were proposed with the objective to minimize the total tardiness and the maximum lateness, respectively. A tabu search algorithm to obtain solutions that give the minimum weighted tardiness is presented by Logendran et al. [31]. De Paula et al. [12] and Rocha et al. [40] proposed, re-

spectively, variable neighborhood search and GRASP algorithms to minimize the makespan added to the weighted total tardiness. In [40] also is developed a branch-and-bound algorithm for the same problem. For minimizing the total tardiness, an simple iterated greedy heuristic is presented by Lin et al. [29].

In this paper, we focus on an UPMS problem with machine and job sequence dependent setup times such that the total Earliness and Tardiness (E/T) penalties is minimized. We deal a general case of this problem in which the jobs have distinct due dates and machine idle times are permitted. The assumption of no inserted idle times is inconsistent with the earliness/tardiness criterion because earliness is an irregular performance measure [6]. Criteria related with E/T are very important in Just-in-Time (JIT) production environments. In JIT production, jobs should be completed at times as close as possible to the due dates: both earliness and tardiness should be discouraged. This is due to the fact that an early job may result in inventory carrying cost, such as opportunity cost of the money invested in inventory, storage and insurance costs and deterioration. Contrarily, a tardy job may result in customer dissatisfaction, contract penalties, loss of sale and loss of reputation. Therefore, the criterion involving both E/T costs has received significant attention recently [27].

Many studies considering both E/T penalties deal with single machine problems [6], [25], [45], [49], [5]. Only a few works have investigated problems with parallel machines. Biskup and Cheng [8] addressed the identical PMS problem with the objective of minimizing the earliness, tardiness and completion time penalties. They showed that the problem is NP-hard and developed a efficient heuristic. Sivrikaya and Ulusoy [44] developed two genetic algorithms to tackle the PMS problem with E/T penalties in which the jobs have distinct due dates. These authors consider sequence-dependent setups. Bank and Werner [7] considered UPMS regarding release date as well as common due date. They proposed constructive and local search heuristics for minimizing the weighted sum of E/T penalties. Later, Kedad-Sidhoum et al. [22] proposed efficient lower bounds for identical PMS problem with distinct due dates and the E/T costs. They also propose a simple local search algorithm in order to derive upper bounds. M'Hallah and Al-Khamis [35] addressed a PMS problem with distinct due dates regarding allowable machine idle time. They developed a mixed-integer model and proposed hybrid heuristics, based on genetic algorithm and simulated annealing, to minimize the total weighted E/T. Vallada and Ruiz [46] studied UPMS problem with machine and job sequence dependent setup times with the objective of minimizing the total weighted E/T. The idle time is allowed in their research. Kayvanfa et al. [21] studied the the UPMS problem with sequence dependent setup times and with the objective of minimizing total weighted E/T, makespan as well as jobs cost compressing and expanding depends on the amount of compression/expansion. They also assumed that jobs due dates are distinct and machine idle time is not allowed. To solve medium-to-large size instances, they employed heuristic and metaheuristic approaches.

In this paper we develop three heuristics based on GRASP methodology [14]. The first heuristic is a basic GRASP algorithm which consists of two phases: a

solution construction phase, which randomly constructs a greedy solution, and an improvement phase, which uses that solution as an initial starting point. The second heuristic is a hybridization of GRASP with Path Relinking [19]. In the third heuristic we use an ILS heuristic [32] as improvement procedure of GRASP.

GRASP and ILS are metaheuristic algorithms that have been applied with success to solve a variety of combinatorial optimization problems [15], [32]. Path Relinking is a search intensification procedure that explores paths in the neighborhood solution space connecting two good-quality solutions. The hybridization of PR and GRASP adds memory mechanisms to GRASP.

The remainder of this paper is organized as follows. The definition of the examined UPMS problem and the Mixed Integer Programming model formulation are presented in Section 2. We describe in detail the proposed heuristics in In Section 3. Section 4 discusses the computational results. Finally, Section 5 concludes the paper and provides some fruitful directions for future research.

2 Problem Statement and Mathematical Model

The UPMS problem examined in this paper is stated as follows. There is a set $J = \{1, ..., n\}$ of n that have to be processed on exactly one machine out of a set $M = \{1, ..., m\}$ of m parallel machines. Each machine is continuously available and can process at most one job at a time. No job preemptions are allowed. Each job j becomes available at time zero, has a processing time $p_{j,k}$ on machine k, a due date d_j and earliness (α_j) and tardiness (β_j) penalties. Between the processing of two consecutive jobs i and j on machine k is considered a sequence dependent setup time $s_{i,j,k}$. We do not consider setup times before processing the first job on a machine. In this problem the occurrence of machine idle time is allowed. Idle time on a machine may be required to complete a job on its due date, avoiding earliness. The objective of the problem is to determine a feasible schedule so that the total earliness and tardiness penalties of the jobs is minimized. For a schedule s, the minimized criterion is computed as:

$$f(s) = \sum_{j=1}^{n} (\alpha_j E_j + \beta_j T_j) \tag{1}$$

where, $E_j = max\{0, d_j - C_j\}$ is the earliness of job j and $T_j = max\{0, C_j - d_j\}$ is the tardiness of job j, with C_j being the completion time of job j. Note that, a job j is early when it completes before its due date ($C_j < d_j$). Similarly, when a job j ends after its due date ($C_j > d_j$), it has tardiness. The penalties incurred on earliness and tardiness are determined by $\alpha_j E_j$ and $\beta_j T_j$, respectively.

Following the three field notation of Graham et al. [20], the problem we study can be denoted as $R/s_{ijk}/\Sigma(\alpha_i E_i + \beta_i T_i)$, where R is for the unrelated parallel machine environment, s_{ijk} is machine and job sequence dependent setup times and $\Sigma(\alpha_i E_i + \beta_i T_i)$ is the objective function.

We provide a Mixed Integer Programming (MIP) model for the UPMS problem with sequence dependent setup times. The model is based on the MIP model pre-

sented by Morabito et al. [4] for the total earliness and tardiness minimization. The model uses a dummy job 0 to mark the beginning and end of a sequence of jobs on each machine. The model involves the following decision variables:

$C_{i,k}$ = Completion time of job i at machine k

E_i = Earliness of job i

T_i = Tardiness of job i

$$x_{i,j,k} = \begin{cases} 1, \text{ if job } i \text{ precedes job } j \text{ on machine } k, \\ 0, \text{ otherwise.} \end{cases}$$

The mathematical model is:

$$\min \sum_{i=1}^{n} (\alpha_i E_i + \beta_i T_i) \tag{2}$$

$$\sum_{k=1}^{m} \sum_{\substack{i=0 \\ i \neq j}}^{n} x_{i,j,k} = 1, \qquad j = 1, ..., n \tag{3}$$

$$\sum_{j=1}^{n} x_{0,j,k} \leq 1, \qquad k = 1, ..., m \tag{4}$$

$$\sum_{\substack{i=0 \\ i \neq h}}^{n} x_{i,h,k} - \sum_{\substack{j=0 \\ j \neq h}}^{n} x_{h,j,k} = 0, \qquad h = 1, ..., n, k = 1, ..., m \tag{5}$$

$$C_{0,k} = 0, \qquad k = 1, ..., m \tag{6}$$

$$C_{j,k} \geq C_{i,k} - M + (p_{j,k} + s_{i,j,k} + M) x_{i,j,k}, \quad i = 0, ..., n, j = 1, ..., n, k = 1, ..., m \tag{7}$$

$$E_i \geq d_i - C_{i,k}, \qquad i = 1, ..., n, k = 1, ..., m \tag{8}$$

$$T_i \geq C_{i,k} - d_i, \qquad i = 1, ..., n, k = 1, ..., m \tag{9}$$

$$T_i \geq 0, E_i \geq 0, \qquad i = 1, ..., n \tag{10}$$

$$x_{i,j,k} \in \{0, 1\}, \qquad i, j = 0, ..., n, k = 1, ..., m. \tag{11}$$

The objective function (2) is to minimize the the total weighted earliness/tardiness penalties. Constraint set (3) ensures that every job j is assigned to exactly one machine and has exactly one predecessor. The constraints (4) limit the number of successors of the dummy job 0 to a maximum of one on each machine k. Constraints (5) ensure that every job h has exactly one successor, except for the dummy job which establishes the beginning and end of a job sequence on a machine k. For the job 0, constraints (6) state that the completion time of this job on each machine is equal to zero. Constraint set (7) is to control the completion times of the jobs at the machines. If a job j is assigned to machine k after job i (i.e., $x_{i,j,k} = 1$), its completion time $C_{j,k}$ must be greater than the completion time of i, $C_{i,k}$, plus

the setup time between i and j and the processing time of j. If $x_{i,j,k} = 0$, then the big constant M renders the constraint redundant. Constraints (8) and (9) define the earliness and tardiness of each job, respectively. Finally, constraint set (10) identifies the non-negativity conditions and set (11) defines the binary variables.

3 Proposed Heuristic Algorithms

In this work, to obtain near-optimal solutions of the $R/s_{ijk}/\Sigma(\alpha_i E_i + \beta_i T_i)$ problem, we develop tree heuristic algorithms based on the Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic. GRASP, proposed originally by Feo and Resende [14], is a multistart (iterative) two-phase method basically consisting of a solution construction phase and an improvement phase. The construction phase randomly builds a greedy solution step by step, adding elements to a partial solution. When a feasible solution has been built, its neighborhood is explored in a local search phase until a local optimum is found. The best solution produced after a given pre-specified number of iterations (or termination criterion) is returned as the output.

GRASP algorithm is easily adaptable and has been successfully applied for several NP-hard combinatorial optimization problems [15]. In this work we first develop an adaptation of the basic GRASP algorithm to $s_{ijk}/\Sigma(\alpha_i E_i + \beta_i T_i)$ problem. Then we use the Path Relinking (PR) technique to improve the GRASP performance. This technique is used as an intensification strategy to combine the best solutions obtained in the iterative process. We also propose an hybrid heuristic which combines the GRASP algorithm with the Iterated Local Search (ILS) metaheuristic [32]. ILS is used as a substitute of the standard local search in the GRASP algorithm. ILS is an iterative algorithm that at every iteration applies perturbations to local optimum solutions and the resulting perturbed solutions are then submitted to a local search.

The three developed heuristics are named basic GRASP, GRASP+PR and GRASP+ILS+PR, respectively. In the next subsections, first, we describe the representation and evaluation of a feasible solution. Then we describe each phase of the GRASP algorithm (construction and improvement), the PR technique and the ILS local search algorithm.

3.1 Representation and evaluation of solutions

A solution of the $R/s_{ijk}/\Sigma(\alpha_i E_i + \beta_i T_i)$ problem is represented by m linked lists of jobs (one per machine). Each list represents the processing order of the jobs assigned to a machine. For example, a solution for an instance with $n = 6$ jobs and $m = 2$ machines is represented by $s = [s_1, s_2] = [[1, 4, 6], [2, 3, 5]]$, where $s_1 = [1, 4, 6]$ and $s_2 = [2, 3, 5]$ represent the processing order of the jobs assigned to the machines 1 and 2, respectively.

To compute the objective function $f(s)$ of a given solution s, first, the optimal starting times of the jobs are calculated. In the calculation of these times, the occurrence of machine idle time can be allowed. That is, a machine can be idle until the processing of the next job is started. Szwarc and Mukhopadhyay [45]

and Wang and Yen [49] proposed optimal timing algorithms to compute the optimal starting times of the jobs on single machine scheduling. In this work, we adapt these algorithms for the parallel machine case. The implemented algorithm decides the optimal starting time and completion time according to the corresponding due date for each job. The goal of the optimal algorithm is to minimize the total earliness and tardiness penalties for a solution previously determined.

In order to better understand the machine idle time insertion, we make use of an example problem with six jobs and two machines ($n = 6$, $m = 2$). Assume that the due dates of jobs $1, ..., 6$ are $d_1 = 7$, $d_2 = 10$, $d_3 = 18$, $d_4 = 20$, $d_5 = 27$ and $d_6 = 30$, respectively. Let $s = [[1, 4, 6], [2, 3, 5]]$ a solution for this instance. A optimum schedule is showed in Figure 1. In this schedule, the sequence of jobs on the first machine (M1) is $[1, 4, 6]$ and the sequence of jobs on the second machine (M2) is $[2, 3, 5]$. Not that, all jobs end exactly on their due date. It is straightforward to see that all machines have idle times. For example, there is an idle time of 3 time units on machine M2 between the completion time of the job 2 (after the setup time $s_{2,3,2}$) and the beginning of job 3. In this example, the optimal starting times of the jobs $1, ..., 6$ are 3, 4, 15, 14, 22 and 25, respectively. We can note that machine idle times are necessary to avoid job earliness. For the same instance, Figure 2 shows a schedule that does not allow idle times. In this schedule earliness penalties are generated because all the jobs complete before their due dates.

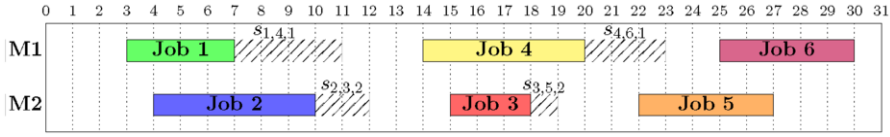

Fig. 1. Two machine schedule: Idle time allowed

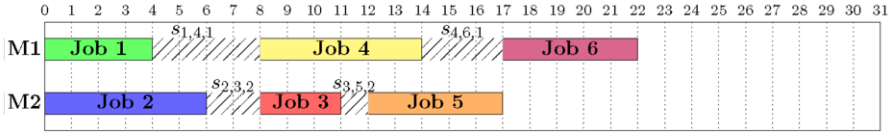

Fig. 2. Two machine schedule: No idle time allowed.

3.2 Solution construction phase

In the construction phase of GRASP algorithm, a solution $s = [s_1, ..., s_m]$ is generated iteratively, where s_k is the sequence of jobs on machine k. Starting with an empty solution ($s_k = \emptyset, \forall k = 1, ..., m$), at each iteration, a job j' is selected uniformly at random from a Restrict Candidate list RC and added to exactly one machine k' (or sequence s'_k). The list RC is formed as follows. First, each unscheduled job j is temporarily assign to machine k that produces the lowest value of the objective function (the total earliness and tardiness penalties of the jobs of the already scheduled jobs). Then, the jobs are arranged according to the produced value, forming the candidate list C of unscheduled jobs. The Restricted Candidate

list RC is formed by the jobs of C that have the best values. The size of RC is $r = max(1, \alpha \times |C|)$, where $\alpha \in [0, 1]$ is the parameter that controls the amounts of greediness and randomness in the construction algorithm. At each construction iteration, the selected job j' is removed from C and the list of unscheduled jobs is reordered. The algorithm to construct a greedy randomised solution ends when all jobs are assigned to machines, i.e, when $C = \emptyset$. At this point a complete solution s is returned. The complete pseudocode of the constructive procedure can be found in Algorithm 1.

Note that, $\alpha = 0$ ($|RC| = 1$) corresponds to a greedy construction in which the first job of C is always selected. $\alpha = 1$ ($|RC| = |C|$) produces a random construction in which a job is chosen randomly from C.

The way of ordering the unscheduled jobs is also used in the constructive heuristic DJASA (Dynamic Job Assignment with Setups Resource Assignment) proposed by Ruiz and Andrés [42]. DJASA is a deterministic greedy heuristic in which the first job of the ordered list is always selected, i.e. the job that provides the least increase in the objective function value.

Algorithm 1 GreedyRandomizedConstruction(α)

```
Let s_k = ∅, ∀k = 1, ..., m;
Add all jobs to a list of unscheduled jobs C (candidate list);
while C ≠ ∅ do
    for For every pending job j ∈ C do
        Temporarily assign job j to machine k that produces the lowest value of the objective function
        (the corresponding machine of job j is k);
        Let g(j) the objective function value for the obtained partial solution;
    end for
    Arrange the jobs in C in increasing order of the function g;
    Let RC the restrict candidate list formed by the first max(1, α × |C|) jobs of C;
    Select job j' at random from RC;
    Assign job j' to the corresponding machine k' (s_k' = s_k' ∪ {j'});
    Remove job j' from C;
end while
Return the obtained solution s;
```

3.3 Improvement phase

Each solution s built in the constructive phase is the starting point for a Local Search procedure in which we try to improve the solution. The LocalSearch method implemented in this work is based on neighborhood search. This method generates new solutions (neighbor solutions) through job insertions made in the current solution s. An insert move generates a new solution by removing a job from its original position u and inserting it into position v of the same or different machine.

The neighborhood contains all the solutions reached through single moves made in the current solution. In this neighborhood a solution that is better than the current solution is picked up. The chosen solution becomes a new solution (or current) and the process continues until a local minimum is reached. The pseudocode of the Local Search procedure can be seen in Algorithm 2.

The insertion neighborhood of a solution s, with $s_k \neq \emptyset$, $\forall k = 1, ..., m$, has size $(n^2 - n + m)$ if m is even or size $(n^2 - m)$ if m is odd. Consider the following example with four jobs and two machines. The neighborhood of solution s = [[2, 1], [3, 4]] is

formed by 14 solutions: s_1 = [[1, 2], [3, 4]], s_2 = [[1], [2, 3, 4]], s_3 = [[1], [3, 2, 4]], s_4 = [[1], [3, 4, 2]], s_5 = [[2], [1, 3, 4]], s_6 = [[2], [3, 1, 4]], s_7 = [[2], [3, 4, 1]], s_8 = [[3, 2, 1], [4]], s_9 = [[2, 3, 1], [4]], s_{10} = [[2, 1, 3], [4]], s_{11} = [[2, 1], [4, 3]], s_{12} = [[4, 2, 1], [3]], s_{13} = [[2, 4, 1], [3]] and s_{14} = [[2, 1, 4], [3]].

Algorithm 2 LocalSearch(s)

```
1: Determine the insertion neighborhood of solution s, N(s);
2: Let s' the better solution in N(s);
3: if f(s') < f(s) then
4:     s ← LocalSearch(s');
5: end if
6: Return s;
```

3.4 Intensification phase with Path Relinking

The Path Relinking (PR) technique is an intensification strategy proposed by Glover [19]. This technique generates new solutions by exploring trajectories that connect high-quality (elite) solutions previously produced during the search. The PR needs a pair of solutions, say s_o (initial solution) and s_g (guiding solution), $s_o \neq s_g$. A path that links s_o to s_g is generated by applying neighborhood moves to the initial solution, which progressively introduces attributes from the guiding solution. At each step, all movements that incorporate attributes of the guiding solution are analyzed and the movement that best improves (or least deteriorates) the current solution is chosen to be the next intermediate solution of the path.

Resende and Ribeiro [39] describe alternatives that have been considered in recent implementations of PR between two input solutions s_o and s_g. In this work, we implement the PR variant called Mixed Path Relinking. Instead of starting from a solution s_o and gradually transforming it into the solution s_g, this variant performs one step from s_o to s_g, obtaining an intermediate solution s_1. Then s_g becomes the initial solution and s_1 the guiding solution, obtaining a new intermediate solution s_2. In the next step of the procedure s_1 becomes the initial solution and s_2 the guiding solution, obtaining s_3 and so on. This process is executed until both paths joint in the middle. The main advantage of this strategy is that it explores deeply neighborhoods of both input solutions.

In our implementation, the initial solution s_o is the solution returned by the local search and the guiding solution s_g is a solution selected at random from a set E of elite solutions. This set represents the pool of the best different solutions found by the GRASP algorithm.

To generate neighbor solutions we use swap and insertion moves. Swap move is used to put a job of s_o into position occupied in guiding solution. For example, if a job j has different positions in the solutions s_o and s_g, and j has position v in s_g, then job j is swapped with job j_v, where j_v is the job that occupies the position v in solution s_o. Swap moves are made only for jobs ($\in s_o$) that occupy different positions in solutions s_o and s_g.

The insertion move is used to removing a job from a machine k and inserting it in other machine k'. If a machine k (in s_o) has a larger number of jobs compared

to the same machine in s_g, then the last job of machine k is removed and inserted in a machine k' with a smaller number of jobs. With insertion moves we obtain sequences with the same sizes as in the guide solution.

Figure 3 shows an example in which five neighbor solutions are generated from $s_o = [[9, 6, 7, 8, 3], [12, 1, 10, 4], [2, 5, 11]]$ considering $s_g = [[9, 6, 2, 8], [3, 12, 10, 1, 4], [7, 5, 11]]$ as guiding solution. In this example, six jobs of s_o have different positions in s_o and s_g, they are 7, 3, 12, 1, 4 e 2. The following swaps are made: (7, 2), (3, 12), (12, 1) and (1, 4). Swaps concerning jobs 4 and 2 are not made. The job 4 should be swapped with the job that occupy the fifth position in machine 2. This exchange is impossible since in machine 2 there is no such position. The job 2 should be swapped with the job 7, but this swap is not necessary because it will yield a solution already analyzed. In this example, we can note that job 3 of the machine 1 was inserted into the machine 2. This move is made because the machine 1 in solution s_o has a larger number of jobs in comparison to the same machine in solution s_g.

In PR process the neighbor solution with minimum objective function value is selected for the next step. In the example of Figure 3, the solution with $f = 1396$ will be selected as intermediate solution.

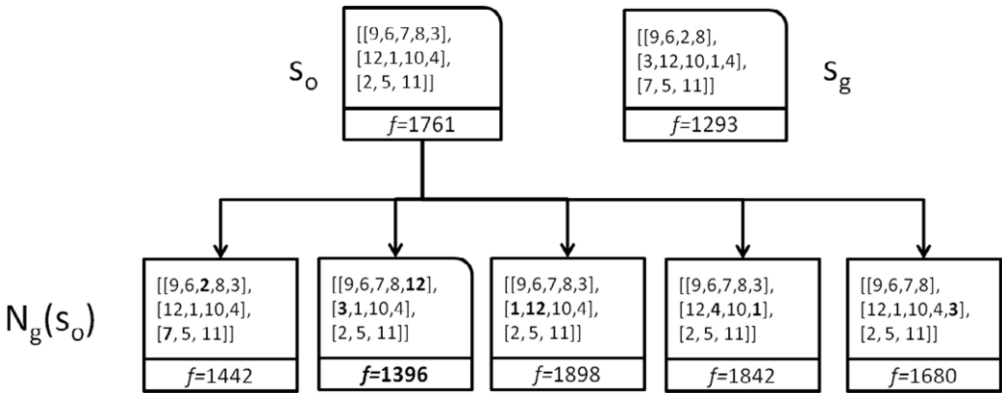

Fig. 3. Neighbor solutions obtained in one step of the Path Relinking procedure

3.5 GRASP+PR algorithm

A general pseudocode description of the GRASP+PR algorithm is presented in Algorithm 3. The algorithm has three parameters to be defined, the randomness parameter α used in the construction phase, the maximum size of the elite set (e_{size}) used in the Path Relinking procedure and the stop condition (*Stopping*).

The algorithm starts initializing the elite set E as empty. During each iteration of the algorithm, a greedy randomized solution s is constructed and improved by the local search procedure, resulting in a local minimum s_o. If the elite set E has at least 2 elements, then the Mixed Path Relinking is applied between s_o and an elite solution s_g randomly chosen from E. The solution returned by the PR procedure is used to update the elite set E. The best elite solution is returned by the GRASP+PR algorithm.

Algorithm 3 GRASP+PR(α, e_{size}, *Stopping*)

```
1: E ← ∅; i ← 1;
2: while Stopping do
3:     s ← GreedyRandomizedConstruction(α);
4:     s_o ← LocalSearch(s);
5:     if i ≥ 2 then
6:         s_g ← Randomly select a solution from E;
7:         s_o ← MixedPathRelinking(s_o, s_g);
8:         E ← EliteSetUpdate(s_o, E, e_size);
9:     else
10:        E ← EliteSetUpdate(s_o, E, e_size);
11:    end if
12:    i ← i + 1 ;
13: end while
14: Return the best solution s* ∈ E;
```

A pseudocode for the elite set update subroutine can be seen in Algorithm 4. The set E stores at most e_{size} high-quality solutions. If the elite set E is not full, a solution s is added to E if $s \notin E$. When the elite set is full, a solution s is added to E if $s \notin E$ and it is better that at least one solution in E. Among all elite solutions having objective function no better than that of s, the solution s' most similar to s is selected to be removed from the elite set. s is added in E, in place of the solution s'. The similarity between two solutions is determined by the number of job occupying the same positions.

Algorithm 4 EliteSetUpdate (s, E, e_{size})

```
1: if |E| = e_size then
2:     if f(s) ≤ max{f(s')|s' ∈ E} and s ∉ E then
3:         Replace the solution s' ∈ E most similar to s among all elements with objective function worst
           than s;
4:     end if
5: else
6:     if s ∉ E then
7:         E ← E ∪ {s};
8:     end if
9: end if
10: return E;
```

3.6 ILS algorithm as improvement phase

An efficient hybrid algorithm proposed here brings together the components of GRASP, ILS and PR. This algorithm, called GRASP+ILS+PR, is similar to GRASP+PR. The difference is in the improvement procedure (Local Search). GRASP+ILS+PR uses the Iterated Local Search (ILS) heuristic in the improvement phase.

ILS [32] is a simple and generally applicable heuristic that iteratively applies local search to modifications (perturbations) of a current solution s. In this work the ILS algorithm is run until I_{ILS} consecutive perturbations without improvements are performed. The perturbation is always performed on the best current solution s^* of a given iteration (acceptance criterion). In Algorithm 5 is showed the pseudocode of the implemented ILS algorithm. This algorithm has three input parameters, the solution s to be improved, the level of perturbation d and I_{ILS} used as stopping criterion.

The Perturbation method used in ILS is based on two stages, *destruction* and *construction*, as in the Iterated Greedy algorithm proposed by Ruiz and Stützle [43].

Algorithm 5 ILS(s, d, I_{ILS})

```
1: s* ← LocalSearch(s);
2: i ← 0;
3: while i < I_ILS do
4:     s ← Perturbation(s*, d);
5:     s' ← LocalSearch(s);
6:     if f(s') < f(s*) then
7:         s* ← s';
8:         i ← 0;
9:     end if
10:    i ← i + 1;
11: end while
12: Return s*;
```

In the destruction stage, d jobs are randomly removed from the current solution s^* obtaining a partial solution s_p. The construction stage is a greedy procedure in which the previously removed jobs are reinserted into the better positions of the partial solution. The implemented Perturbation method is shown in Algorithm 6. This Algorithm returns the best complete solution generated in the construction stage.

Algorithm 6 Perturbation(s, d)

```
1: s_p ← s; R ← ∅;
2: for i = 1 to d do
3:     s_p ← Remove at random a job j from s_p;
4:     R ← R ∪ {j};
5: end for
6: for each job j ∈ R do
7:     s_p ← best solution obtained after inserting job j in all possible positions of s_p;
8: end for
9: Return s_p;
```

4 Computational Experiments

In this work, we analyze the efficiency of the three developed heuristics: GRASP, GRASP+PR and GRASP+ILS+PR. All algorithms were coded in Java 1.6, using IDE Eclipse 3.5.1 compiler and run on an Intel Core 2 Quad CPU Q9550, 2.83GHz, with 6GB of RAM running under Windows 7, 64 bits OS. Only a single thread was used in the experiments.

The input parameters values of the algorithms were selected after some preliminary experiments. Best results were achieved by using the following parameters. Parameter that controls the amounts of greediness and randomness in the construction phase: $\alpha = 0.1$. Maximum size of the elite set used in the Path Relinking intensification: $e_{size} = \frac{n}{15}$, where n is the number of jobs. Stopping criterion of the ILS algorithm: $I_{ILS} = 50$ (number of iterations without improvements). Level of perturbation used in ILS algorithm: $d = 2$, $d = 4$ and $d = 10$, for small, medium and large instances, respectively.

All the heuristic algorithms (GRASP, GRASP+PR and GRASP+ILS+PR) were run with the same *stopping criterion* which is based on an amount of CPU time. This time is giving by $(n \times m \times 50)$ milliseconds, where m is the number of machines. Therefore, the computational effort increases as the size of the considered instance increases. In this way, we assign more time to larger instances that are naturally

more time consuming to solve.

The computational tests were divided in three experiments. In the first, the heuristics are tested using small instances. We compared the obtained solutions with optimal results obtained by the MIP model fort the considered problem (presented in Section 2) and solved via ILOG-IBM CPLEX 12.2. In the second experiment, the heuristics are tested using medium-to-large instances. The third experiment evaluates the probability distribution of the running time of the heuristics by using time-to-target plots.

The obtained results are analyzed by using the *Relative Percentage Deviation* (RPD) which is considered as the response variable for the experiments. The RPD is computed for each instance according to the following expression:

$$RPD\% = 100 \times \frac{f_{algorithm} - f_{best}}{f_{best}} \tag{12}$$

where f_{best} is the best known solution (objective function value) obtained among all the algorithms after all the experiments carried out throughout the paper, and $f_{algorithm}$ is the solution obtained with a given algorithm. We run five replicates of each algorithm.

4.1 Problems Instances

Computational experiments are performed on 400 randomly generated instances of the problem. According to the number of jobs (n) and number of machines (m), the test problems are classified in three sets, named I, II and III. The Set I contains small instances where n and m are assume in $\{8, 9, 10\}$ and $\{3, 5\}$, respectively. In the Set II there are medium instances with $n \in \{20, 30\}$ and $m \in \{3, 5\}$. The Set III is formed by large instances where $n \in \{50, 60, 70, 80, 90, 100\}$ and $m \in \{10, 15, 20, 25, 30\}$. For each couple (n, m), 10 instances were generated.

The scheme for instance generation is similar to that presented by Lee and Pinedo [26]. The processing time $p_{i,k}$ of job i on machine k is an integer randomly generated from uniform distribution over the interval $[50, 100]$. The setup times are uniformly distributed over the range $\left[\frac{2}{3}\eta\overline{p}, \frac{4}{3}\eta\overline{p}\right]$, where $\overline{p}$ is the mean of the processing times and $\eta = 0.25$. The earliness and tardiness penalties, α_i and β_i, are uniformly distributed on $[1, 100]$. The due dates d_j of jobs are uniformly distributed over the interval $[(1-R)\overline{d}, \overline{d}]$ with probability τ, and uniformly distributed over the interval $[\overline{d}, ((C_{max} - \overline{d})R) + \overline{d}]$ with probability $(1-\tau)$, where $\overline{d} = C_{max}(1-\tau)$ is the median of due dates, and C_{max} is a estimator for the makespan computed in the following way:

$$C_{max} = \frac{n}{m}\left(\overline{p} + \overline{s}\left(0, 4 + \frac{10m^2}{n^2} - \frac{\eta}{7}\right)\right) \tag{13}$$

where $\overline{s}$ is the average time of setup times. The two parameters τ and R are the tardiness factor and the dispersion range of due dates, respectively. These parameters have been assumed as follows: $\tau = 0.3$, $R = 0.25$.

4.2 Experiment 1: Comparison with optimal solutions

In this first experiment, we use the Set I of small instances. The MIP model for these instances were solved by ILOG-IBM CPLEX 12.2 software. This software was able to obtain the optimal solutions for all the instances of Set I spending a low computational time.

The obtained results in this experiment are reported in Table 1. The first column of this Table shows the sizes $n \times m$ of the tested instances. The second and third columns present the average CPU time (in seconds) required by the CPLEX and the heuristics algorithms. The others columns shows the average RPD for the proposed heuristics. We can see that GRASP+PR and GRASP+ILS+PRS perform better than the basic GRASP algorithm, except for instances of size 8×3. However, the solutions obtained by the GRASP algorithm were very near to the optimal solutions. In the Table, the best RPD values are in boldface. Furthermore, the mean times for the heuristic algorithms were less than that of CPLEX, for all the instances.

Table 1
Average CPU times and RPD for the proposed heuristic algorithms (small instances)

$n \times m$	Mean CPU time (s)		**GRASP**	**GRASP+PR**	**GRASP+ILS+PR**
	CPLEX	Heuristics			
8×3	8.47	1.20	**0.00**	**0.00**	**0.00**
8×5	8.03	2.00	0.18	**0.00**	**0.00**
9×3	9.44	1.35	0.61	**0.00**	0.35
9×5	12.54	2.25	0.25	0.08	**0.01**
10×3	7.54	1.50	0.14	**0.00**	**0.00**
10×5	6.54	2.50	1.16	1.04	**0.83**
Average	8.76	1.80	0.39	**0.18**	0.19

4.3 Experiment 2: Tests with medium and large instances

In the second experiment, the developed heuristics were tested on medium and large instances.

For the 40 medium instances, the heuristics are also compared with CPLEX solver. This solver was applied to solve the MIP model with a threshold CPU time of three hours for each medium instance. That is, if after the established time no optimal solution is obtained, the best current solution (upper bound) is returned by CPLEX.

The solver, within the established time, is not able to obtain the optimal solution for all the medium instances. In Table 2, results for medium instances are shown for all the evaluated heuristics including the CPLEX solver. This Table presents the average RPD for the compared methods. We can see that the performances of GRASP+PR and GRASP+IL+PR are better than GRASP. The GRASP+ILS+PR heuristic performs slightly better than GRASP+PR. We can note also that the RPD values of CPLEX are much greater than those of the heuristic algorithms. Such poor results for the optimal solver suggest that heuristics may be more appropriate for medium and large instances.

In order to analyze the heuristic algorithms when the size of the instance is increased, experiments are also carried out using the large set of instances. In this

Table 2
Average RPD for the proposed heuristic algorithms (medium instances)

$n \times m$	CPLEX	GRASP	GRASP+PR	GRASP+ILS+PR
20x3	19.38	0.76	0.19	**0.16**
20x5	29.63	2.22	0.78	**0.59**
30x3	66.60	1.42	**0.98**	**0.98**
30x5	111.54	2.29	0.94	**0.74**
Average	56.78	1.67	0.72	**0.62**

case MIP model is not tested. In Table 3 we can see the results for large instances obtained by all the heuristics, where the 10 instances of each $n \times m$ group have been averaged. This Table reports the mean RPD values. The best results for each $n \times m$ are in boldface. We can see that the GRASP+ILS+PR heuristic shows a very good performance and provides the best RPD values for all the instances. It is noticeable that, by using PR intensification, the performance of the GRASP heuristic improved, on average, 3.7%. Furthermore, by using ILS in the local search phase of GRASP, the results improved 1.7% on average.

Table 3
Mean RPD for the proposed algorithms (large instances)

Instance $n \times m$	GRASP	GRASP+PR	GRASP+ILS+PR
50×10	3.92	0.81	**0.78**
50×15	6.76	1.90	**0.53**
50×20	7.03	2.08	**0.49**
50×25	10.02	3.30	**0.75**
50×30	8.15	2.22	**1.18**
60×10	3.39	1.18	**0.50**
60×15	6.02	1.85	**0.61**
60×20	5.24	0.00	**0.00**
60×25	7.34	2.29	**0.41**
60×30	11.00	3.72	**0.49**
70×10	3.82	2.08	**0.45**
70×15	4.47	1.66	**0.21**
70×20	6.17	2.35	**0.48**
70×25	7.40	1.98	**0.36**
70×30	7.89	2.64	**0.66**
80×10	2.95	1.71	**0.79**
80×15	4.18	2.04	**0.41**
80×20	5.51	2.14	**0.56**
80×25	6.99	2.30	**0.31**
80×30	7.72	2.96	**0.36**
90×10	3.61	2.12	**0.42**
90×15	4.44	2.59	**0.08**
90×20	5.83	2.51	**0.30**
90×25	5.84	2.16	**0.16**
90×30	6.80	2.30	**0.20**
100×10	3.39	2.59	**0.08**
100×15	3.23	1.64	**0.33**
100×20	5.41	2.42	**0.37**
100×25	6.34	2.82	**0.08**
100×30	6.05	2.28	**0.48**
Average	5.90	2.15	**0.43**

4.3.1 Statistical analysis

In order to validate the results, we apply again an statistical analysis using the RPD measure as response variable. First, to verify if the observed differences between the obtained results are statistically significant, we performed an analysis of variance(ANOVA) [36]. The three main assumptions of ANOVA were checked: normality, homoscedasticity and independence. Since the normality test was not satisfied, we performed the non-parametric Kruskal-Wallis Test. This Test compares

between the medians of the three heuristics to determine if there is a significant difference.The Kruskal-Wallis results (not shown in detail due to reasons of space) indicate that there is statistically significant difference between the obtained results at a 95% confidence level. The Right-Tail Probability was $p - value = 0.00193$ (less than 0.05).

The Kruskal Wallis Test does not specify which algorithms are different. So, we use a non-parametric Multiple Comparisons test to compare each pair of means with a 95% confidence level. The Table 4 shows the result of this test. The first column of this Table shows the pairs of algorithms being compared. The "Difference" column displays the sample mean of the first algorithm minus that of the second. The "+/− Limits" column shows an uncertainty interval for the difference. Any pair for which the absolute value of the difference exceeds the limit is statistically significant at the selected confidence level 95% and is indicated by an (*) in the "Significant" column. In this Table we can see that there are significant differences between all the pairs of algorithms.

The same analysis can be displayed in Figure 4. This Figure shows the means plot and Tukey's Honestly Significant Difference (HSD) intervals at 95% confidence level from the Multiple Comparisons test. Since the confidence interval of GRASP+PR+ILS algorithm does not overlap any of the other intervals, the mean of GRASP+PR+ILS is significantly different than that of the other three algorithms. So we can state that, on average, GRASP+PR+ILS is better than GRASP and GRASP+PR. We can see also that, on average, GRASP+PR is better than GRASP.

The statistical analysis shows that the use of LS and PR improves significantly the results of the basic GRASP algorithm.

Table 4
Multiple Comparisons test

Contrast	**Significant**	**Difference**	**+/- Limits**
GRASP ↔ GRASP+PR	*	3,79	0,31
GRASP ↔ GRASP+ILS+PR	*	5,52	0,31
GRASP+ILS+PR ↔ GRASP+PR	*	-1,73	0,31

4.4 Experiment 3: Run time distributions

In this experiment we use time-to-target (TTT) plots to compare the developed stochastic heuristics by comparing their running time distributions. A TTT plot [2] display the probability that an algorithm will find a solution at least as good as a given target value within a given running time. Basically, to plot the empirical run time distribution of a given stochastic algorithm, a solution target value is fixed and each algorithm is executed T times, measuring the time t_i it takes to find a solution at least as good as the given target solution. For each algorithm, the ith sorted running time t_i is associate to probability $p_i = (i - 1/2)T$. The TTT plot represents

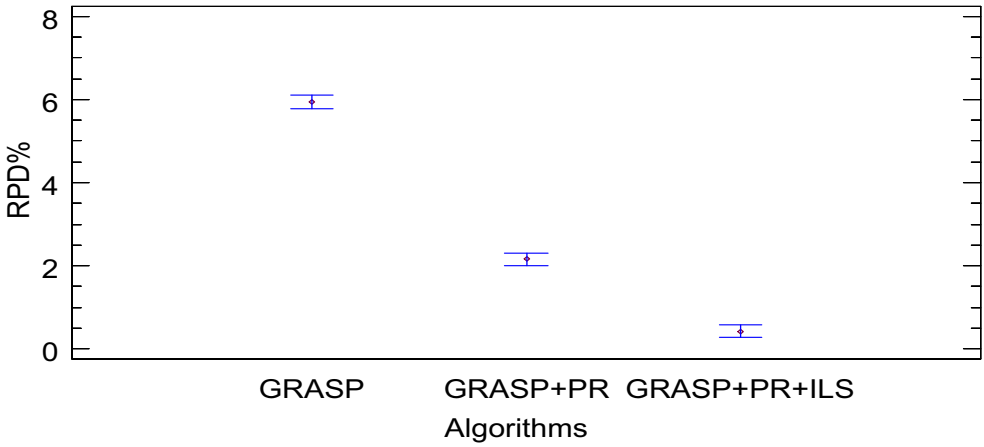

Fig. 4. Means plot and Tukeys HSD intervals with 95% confidence level.

the points (t_i, p_i) for $i = 1, ..., T$.

In this work, to quantify the PR and ILS contribution on the solution space search, we compared the algorithms GRASP, GRASP+PR and GRASP+ILS+PR by producing TTT plots with $T = 50$ independent runs for some representative instances. The target solution is the best solution obtained by the basic GRASP heuristic after $n \times m \times 50$ milliseconds (stopping criterion).

Figure 5 shows the produced TTT plots for a instance with $n = 100$ and $m = 10$. These TTT plots were created by using the a perl program developed by Aiex et al. [1]. The Figure clearly shows that the running times of the GRASP+PR and GRASP+ILS+PR were much smaller than those of the pure GRASP. For example, GRASP+PR algorithm has about 70% chance to hit the solution target value in less than 50 s. With the same probability, GRASP algorithm requires 200% additional time in average (about 150 s) to hit the same target.

This experiment confirms that the additional computational effort produced by the PR and ILS strategies are not only extra weights for the GRASP heuristic. These strategies effectively help to finding better solutions in reduced execution times.

5 Conclusão

This paper has proposed three GRASP heuristics for the UPMS problem with machine and job sequence dependent setup times with the objective of minimizing the total weighted airlines and tardiness. In this problem machine idle time are allowed. A MIP model is also formulated to solve small instances of the problem. The hybrid heuristics, GRASP+PR and GRASP+ILS+PR, combine the basic GRASP scheme with other elements such as Path Relinking (PR) and Iterated Local Search (ILS).

We have performed a comparative study between the proposed heuristics by using small, medium and large instances. According to the extensive experimental and statistical analyses, the proposed GRASP+PR and GRASP+ILS+PR heuristics performed better than the basic GRASP, that is, the results of GRASP are improved

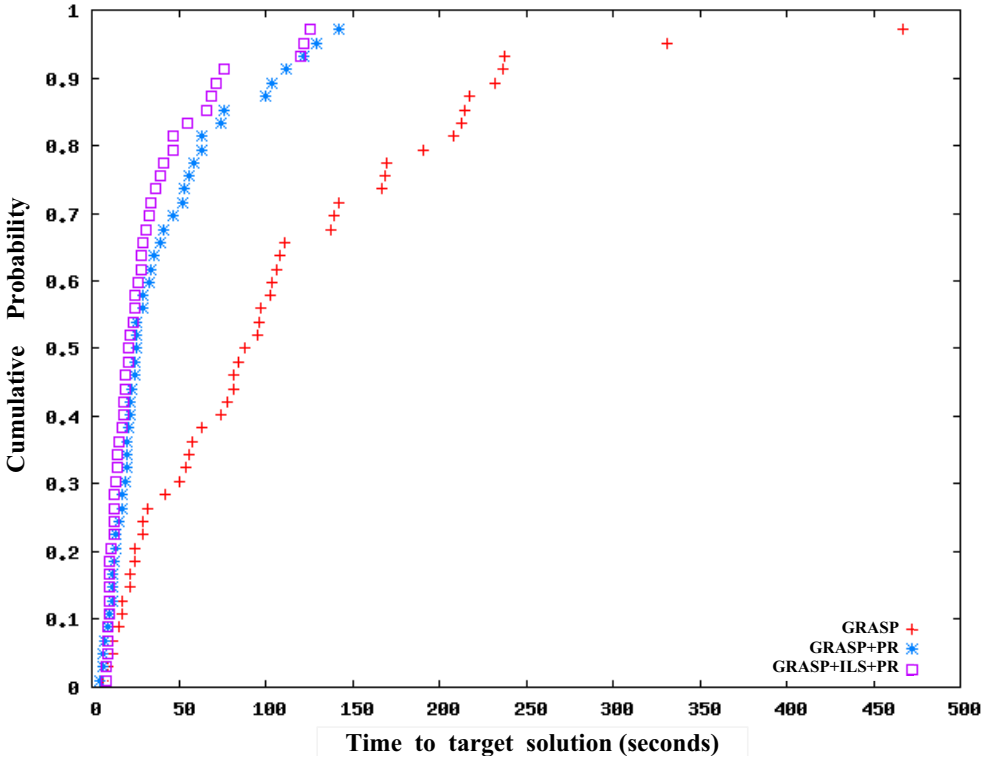

Fig. 5. TTT plots for an instance instance with $n = 100$ ad $m = 10$.

significantly by using PR and ILS.

We analyze also the running time distribution of the heuristics. The effectiveness of PR and ILS in the GRASP run time was also demonstrated by time-to-target plots.

An interesting future research direction is to investigate the use of Variable Neighborhood Descent metaheuristic in the local search phase of ILS.

References

[1] Aiex, R., M. Resende and C. Ribeiro, *Ttt plots: A perl program to create time-to-target plots*, Optimization Letters **1** (2006), pp. 355–366.

[2] Aiex R.M., M. R. C., Resende, *Probability distribution of solution time in GRASP: An experimental investigation*, J. of Heuristics **8** (2002), pp. 343–373.

[3] Allahverdi, A., C. T. Ng, T. C. E. Cheng and M. Y. Kovalyov, *A survey of scheduling problems with setup times or costs*, European Journal of Operational Research **187** (2008), pp. 985–1032.

[4] Arenales, M., R. Morabito, V. A. Armentano and H. Yanasse, "Pesquisa operacional - Modelagem e algoritmos," Elsevier, Rio de Janeiro, 2007.

[5] Arroyo, J., R. Ottoni and A. P. Oliveira, *Multi-objective variable neighborhood search algorithms for a single machine scheduling problem with distinct due windows*, Electronic Notes in Theoretical Computer Science **281** (2011), pp. 5 – 19.

[6] Baker, K. and G. Scudder, *Sequencing with earliness and tardiness penalties: a review*, Operations Research **38** (1990), pp. 22 – 36.

[7] Bank, J. and F. Werner, *Heuristic algorithms for unrelated parallel machine scheduling with a common due date, release dates, and linear earliness and tardiness penalties*, Mathematical and Computer Modeling **33** (2001), pp. 363 – 383.

[8] Biskup, D. and T. Edwin Cheng, *Multiple-machines cheduling with earliness,tardiness and completion time penalties*, Computers & Operations Research **26** (1999), pp. 45 – 57.

[9] Brucker, P., "Scheduling algorithms," Springer Verlag 5th edition, Berlin Heidelberg, 2007.

[10] Chang, P. C. and S. H. Chen, *Integrating dominance properties with genetic algorithms for parallel machine scheduling problems with setup times*, Applied Soft Computing **11** (2011), pp. 1263 – 1274.

[11] Cheng, T. C. E. and C. C. S. Sin, *A state-of-the-art review of parallel machine scheduling research*, European Journal of Operational Research **47** (1990), pp. 271 – 292.

[12] De Paula, M. R., M. G. Ravetti, G. R. Mateus and P. M. Pardalos, *Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighborhood search*, IMA Journal of Management Mathematics **18** (2007), pp. 101 – 115.

[13] Fanjul-Peyro, L. and R. Ruiz, *Iterated greedy local search methods for unrelated parallel machine scheduling*, European Journal of Operational Research **207** (2010), pp. 55 – 69.

[14] Feo, T. and M. Resende, *Greedy randomized adaptive search procedures*, J. of Global Optimization **6** (1995), pp. 109–133.

[15] Festa, P. and M. Resende, *An annotated bibliography of GRASP, Part II: Applications*, International Transactions in Operational Research **16** (2009), pp. 131–172.

[16] Franca, P., M. Gendreau, G. Laporte and F. Müller, *A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times*, International Journal of Production Economics **43** (1996), pp. 79 – 89.

[17] Garey, M. and D. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness," W. H. Freeman, 1979.

[18] Glass, C. A., C. N. Potts and P. Shade, *Unrelated parallel machine scheduling using local search*, Mathematical and Computer Modelling **20** (1994), pp. 41 – 52.

[19] Glover, F., *Tabu search and adaptive memory programing*, Interface in Computer Science and Operational Reaserch (1996), pp. 1–75.

[20] Graham, R., E. Lawler, J. Lenstra and A. Kan, *Optimization and approximation in deterministic sequencing and scheduling: a survey*, Annals of Discrete Mathematics **5** (1979), pp. 287 – 326.

[21] Kayvanfa, V., G. Komaki, Aalaei and A. Zandieh, *Minimizing total tardiness and earliness on unrelated parallel machines with controllable processing times*, Computers & Operations Research **41** (2014), pp. 31 – 43.

[22] Kedad-Sidhoum, S., Y. Solis and F. Sourd, *Lower bounds for the earliness-tardiness scheduling problem on parallel machines with distinct due dates*, European Journal of Operational Research **189** (2008), pp. 1305 – 1316.

[23] Kim, C. and H. Shin, *Scheduling jobs on parallel machines: a restricted tabu search approach*, International Journal of Advanced Manufacturing Technology **22** (2003), pp. 278 – 287.

[24] Kim, D., K. Kim, W. Jang and F. F. Chen, *Unrelated parallel machine scheduling with setup times using simulated annealing*, Robotics and Computer Integrated Manufacturing **18** (2002), pp. 223 – 231.

[25] Lee, C. Y. and J. Y. Choi, *A genetic algorithm for job sequencing problems with distinct due dates and general early-tardy penalty weights*, Computers & Operations Research **22** (1995), pp. 857–869.

[26] Lee, Y. H. and M. Pindedo, *Scheduling jobs on parallel machines with sequence-dependent setup times*, European Journal of Operational Research **100** (1997), pp. 464–474.

[27] Liao, C.-J. and C.-C. Cheng, *A variable neighborhood search for minimizing single machine weighted earliness and tardiness with common due date*, Computers & Industrial Engineering **52** (2007), pp. 404 – 413.

[28] Liaw, C. F., Y. K. Lin, C. Y. Cheng and M. Chen, *Scheduling unrelated parallel machines to minimize total weighted tardiness*, Computers & Operations Research **30** (2003), pp. 1777 – 1789.

[29] Lin, S., L. C.C. and K. C. Ying, *Minimization of total tardiness on unrelated parallel machines with sequence-and machine-dependent setup times under due date constraints*, The International Journal of Advanced Manufacturing Technology **53** (2011), pp. 353 – 361.

[30] Lin, Y. K., M. E. Pfund and J. W. Fowler, *Heuristics for minimizing regular performance measures in unrelated parallel machine scheduling problems*, Computers & Operations Research **38** (2011), pp. 901 – 916.

[31] Logendran, R., B. McDonell and B. Smucker, *Scheduling unrelated parallel machines with sequence-dependent setups*, Computers & Operations Research **34** (2007), pp. 3420 – 3438.

[32] Lourenço, H. R., O. C. Martin and T. Stützle, *Iterated local search*, in: G. Fred and K. Gary, editors, *Handbook of Metaheuristics*, International Series in Operations Research & Management Science **57**, Springer New York, 2003 pp. 320–353.

[33] M., P., J. W. Fowler and J. N. D. Gupta, *A survey of algorithm for single and multiobjective unrelated parallel machine deterministic scheduling problems*, Journal of the Chinese Institute of Industrial Engineers **21** (2004), pp. 230 – 241.

[34] Martello, S., F. Soumis and P. Toth, *Exact and approximation algorithm for makespan minimization on unrelated parallel-machines*, Discrete Applied Mathematics **75** (1997), pp. 169 – 188.

[35] M'Hallah, R. and T. Al-Khamis, *Minimising total weighted earliness and tardiness on parallel machines using a hybrid heuristic*, International Journal of Production Research **50** (2012), pp. 2639 – 2664.

[36] Montgomery, D. G., "Design and Analysis of Experiments," Wiley; 8 edition, New York, 2012.

[37] Piersma, N. and W. van Dijk, *A local search heuristic for unrelated parallel machine scheduling with efficient neighborhood search*, Mathematical and Computer Modeling **24** (1996), pp. 11 – 19.

[38] Pinedo, M. L., "Scheduling: theory, algorithms, and systems," Springer Verlag, New York, 2008.

[39] Resende, M. and C. Ribeiro, *Grasp with path-relinking: Recent advances and applications*, in: T. Ibaraki, K. Nonobe and M. Yagiura, editors, *Metaheuristics: Progress as Real Problem Solvers*, Operations Research/Computer Science Interfaces Series **32**, Springer, 2005 pp. 29–63.

[40] Rocha, P. L., M. G. Ravetti, G. R. Mateus and P. M. Pardalos, *Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times*, Computers & Operations Research **35** (2008), pp. 1250 – 1264.

[41] Rodriguez, F. J., M. Lozano, C. Blum and C. García-Martínez, *An iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem*, Computers & Operations Research **40** (2013), pp. 1829 – 1841.

[42] Ruiz, R. and C. Andrés-Romano, *Scheduling unrelated parallel machines with resource-assignable sequence-dependent setup times*, The International Journal of Advanced Manufacturing Technology **57** (2011), pp. 777 – 794.

[43] Ruiz, R. and T. Stützle, *A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem*, European Journal of Operational Research **177** (2007), pp. 2033–2049.

[44] Sivrikaya, F. and G. Ulusoy, *Parallel machine scheduling with earliness and tardiness penalties*, Computers & Operations Research **26** (1999), pp. 773 – 787.

[45] Szwarc, W. and S. Mukhopadhyay, *Optimal timing schedules in tardiness and earliness single machine sequencing*, Naval Research Logistics **42** (1995), pp. 1109 – 1114.

[46] Vallada, E. and R. Ruiz, *Scheduling unrelated parallel machines with sequence dependent setup times and weighted earlinesstardiness minimization*, in: R. Ríos-Mercado and Y. Ríos-Solís, editors, *Just-in-TimeSystems*, Springer Optimization and Its Applications .

[47] Vallada, E. and R. Ruiz, *Genetic algorithms for the unrelated parallel machine scheduling problem with sequence dependent setup times*, European Journal of Operational Research **3** (2011), pp. 612–622.

[48] Van de Velde, S. L., *Duality-based algorithms for scheduling unrelated parallel-machines*, ORSA Journalon Computing **5** (1993), pp. 192 – 205.

[49] Wang, G. and B. Yen, *Tabu search for single machine scheduling with distinct due windows and weighted earliness/tardiness penalties*, European Journal of Operational Research **142** (2002), pp. 129 – 146.