Computers & Operations Research 38 (2011) 1259–1266

Contents lists available at ScienceDirect

Computers & Operations Research

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

Dominance rules for the parallel machine total weighted tardiness scheduling problem with release dates

Antoine Jouglet [a,*], David Savourey [b]

[a] Heudiasyc, UMR CNRS 6599, Université de Technologie de Compiègne, BP 20529, 60205 Compiègne, France
[b] Ecole Polytechnique, CNRS LIX, F-91128 Palaiseau, France

ARTICLE INFO

Available online 16 December 2010

Keywords:
Scheduling
Parallel machines
Total weighted tardiness
Dominance rules
Possible machines

ABSTRACT

We address the parallel machine total weighted tardiness scheduling problem with release dates. We describe dominance rules and filtering methods for this problem. Most of them are adaptations of dominance rules based on solution methods for the single-machine problem. We show how it is possible to deduce whether or not certain jobs can be processed by a particular machine in a particular context and we describe techniques that use this information to improve the dominance rules. On the basis of these techniques we describe an enumeration procedure and we provide experimental results to determine the effectiveness of the dominance rules.

1. Introduction

In this paper we consider the situation where a set of n jobs $J=\{1,\ldots,n\}$ has to be processed on a set of m identical parallel machines $M=\{1,\ldots,m\}$ and where the objective is to minimize the total weighted tardiness. Associated with each job i is a release date r_i, a processing time p_i, a due date d_i and a weight w_i. A job cannot start before its release date and preemption is not allowed. Machines cannot execute more than one job simultaneously. The tardiness of job i is defined as $T_i=\max(0, C_i-d_i)$, where C_i is the completion time of job i. The problem is to find a feasible schedule with minimum total weighted tardiness. This problem, denoted as $Pm|r_i|\sum w_iT_i$, is strongly NP-hard [25].

The single machine problem has been intensively addressed over the last two decades. For the $1|r_i|\sum T_i$ problem, Chu and Portmann [17] and Chu [15] introduced dominance rules, heuristics, a lower bound and branch-and-bound procedures. Baptiste et al. [7] also proposed a lower bound and a branch-and-bound method using efficient dominance rules. For the $1|r_i|\sum w_iT_i$ problem, Akturk and Ozdemir [2] proposed dominance rules used in a branch-and-bound algorithm [1,2]. Finally, Jouglet et al. [22] studied these two criteria and generalized and improved the above dominance rules which were used in a branch-and-bound method. More recently Jouglet et al. [23] described dominance-based heuristic methods for these problems.

As regards parallel machine problems, several exact methods have been described. Yalaoui and Chu [32,33] described dominance rules, lower bounds and branch-and-bound methods for the general $Pm|r_i|\sum C_i$ problem. More recently Nessah et al. [28] described dominance rules and a branch-and-bound method for the weighted case $Pm|r_i|\sum w_iC_i$. Less attention has been given to the tardiness criteria. Nevertheless, Azizoglu and Kirca [3], Yalaoui and Chu [32] and Shim and Kim [30] solved the problem with equal release dates $Pm||\sum T_i$ using a branch-and-bound algorithm and dominance rules. Moreover, Liaw et al. [26] also proposed a branch-and-bound algorithm using dominance properties for the weighted case considering unrelated parallel machines. Finally, Baptiste [6] showed that the special case $Pm|p_i=p,r_i|\sum T_i$ of the total tardiness problem with release dates for which the processing times of jobs are equal can be polynomially solved. More recently Baptiste et al. [8] proposed several lower bounds for the case with release dates.

Unfortunately, dominance rules from the literature [3,26,30,32] cannot be used to solve the $Pm|r_i|\sum w_iT_i$ problem. Indeed, these dominance rules no longer holds when release dates are considered. In this paper we describe new dominance rules for $Pm|r_i|\sum w_iT_i$ based on solution methods for one-machine problems. We describe in Section 2 the enumeration procedure that was used to solve the problem. In Section 3 we introduce the notion of "possible machines" on which a job can be scheduled, and from this we derive some dominance rules. In addition, we use this notion of "possible machines" to improve most of the dominance rules that we shall describe. In Section 4 we generalize a local dominance rule from the literature concerning the one-machine problem. In Section 5 we recall the one-machine dominance rule based on the notion of partial "dominated sequences" of scheduled jobs, and show how this rule can be used and generalized for the parallel machine case.

* Corresponding author. Tel.: +33 3 44 23 79 39; fax: +33 3 44 23 44 77.
E-mail addresses: antoine.jouglet@hds.utc.fr (A. Jouglet), savourey@lix.polytechnique.fr (D. Savourey).

0305-0548/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2010.12.006

Finally, in Section 6 we present some experimental results which show the interest of the described techniques.

2. Enumeration procedure

Among the most commonly used methods for solving combinatorial problems for which there is no known way of avoiding the huge size of their solution space are so-called enumerative techniques [31]. The idea is to find a way of organizing an enumerated list of solutions, using as much information as possible, so as to eliminate intelligently dominated parts of the solution space as the computation progresses, without having to enumerate these dominated parts [24] explicitly. The utility of the method comes from the fact that only a small part of the solution space actually needs to be enumerated [27]. The remaining solutions are eliminated from consideration through the application of dominance rules that establish that such solutions cannot be optimal or better than the best solution already known.

In this section we describe an enumeration procedure for solving the problem exactly. First we present the notion of a nearly active schedule. We show that the subset of nearly active schedules is dominant for the problem. Then we show that there exists a bijection between the set of nearly active schedules and the set of permutations of $(1,\ldots,n)$. This enables us to represent a solution as a permutation and to search for an optimal sequence of jobs instead of searching for the start times of jobs in an optimal schedule.

Building a schedule means allocating resources to jobs over time while respecting any constraints that might be imposed. Clearly, in order to find an optimal schedule, given that the total weighted tardiness is non-decreasing, it is sufficient to visit the set of semi-active schedules, *i.e.* schedules in which no job can be scheduled earlier without changing the execution sequence of jobs on one of the machines [12]. Because this set contains a lot of obviously non-optimal schedules (for example, schedules where all jobs are scheduled on the same machine even though more than one machine is available), we define the subset of "nearly active schedules" which is dominant for the studied problem. Moreover, we are able to build a bijection between the set of permutations of jobs and the set of nearly active schedules. Thus, enumerating all permutations allows us to find an optimal schedule.

We now introduce the notion of nearly active schedule:

Definition 2.1. A schedule is said to be nearly active if it is semi-active and if no job can be scheduled earlier by processing it as the last job on another machine.

Proposition 2.1. *The set of nearly active schedules is dominant for the total weighted tardiness parallel machines problem.*

Proof. Any active schedule is nearly active and the set of active schedules is dominant for the studied problem [5,18]. □

From here onwards we denote by m_i^S the index of the machine on which job i is processed in schedule S, and by Δ_i^S the date of availability of machine m_i^S before the execution of job i in S.

Proposition 2.2. *There exists a bijection between the set Σ of permutations of $(1,\ldots,n)$ and the set of nearly active schedules.*

Proof. Let S be a nearly active schedule. We define the following order relation on the set of jobs in S: $i \prec j$ if and only if $\Delta_i^S < \Delta_j^S$ or $(\Delta_i^S = \Delta_j^S$ and $m_i^S < m_j^S)$. This defines clearly a total order on jobs of S which can be represented by a permutation on jobs in S.

Conversely, given a permutation σ of jobs, a schedule S is associated to σ using the following scheme: jobs are considered in the order given by σ and are scheduled one by one as soon as possible on the machine available the earliest. Where several machines are available at the same earliest time, choose the one with the smallest index number. The obtained schedule is such that no job can be scheduled earlier without changing some sequence order, because jobs are scheduled as soon as possible. Moreover, suppose there exists a job j that can be processed earlier by placing it after the last job scheduled on some other machine. Then, at the step involving job j, this other machine would have been chosen to process j. Thus, there is no job that can be processed earlier by placing it after the last job schedule on another machine and the schedule is nearly active.

Finally, there is a bijection between nearly active schedules and permutations of jobs. □

An illustration from a nearly active schedule and its associated sequence is provided in Fig. 1. In this example, $r_1 = r_2 = r_3 = r_5 = r_6 = r_7 = 0$ and $r_4 = 3$. Jobs are scheduled in the order of the sequence σ following the procedure described in the proof of Proposition 2.2. Thus, there is an idle time before job 4 which is due to its release date.

The nearly active property is stronger than the semi-active property. In fact, this property also ensures that the load is "well balanced" between machines. A schedule in which all jobs are scheduled on the same machine despite there being several machines available can be semi-active but cannot be nearly active.

From now on, the solution to any given problem will be sought only among the set of nearly active schedules. We use the one-to-one mapping between the set of nearly active schedules and the set of permutations of the jobs' index numbers to enumerate solutions. Rather than searching for the starting times of jobs, we look for a sequence of jobs. To build such a sequence, we use the edge-finding branching scheme (see for instance Carlier [13]) when constructing a solution. This involves ordering jobs ("edges" in a graph representing the possible orderings of jobs): at each node a set of jobs is selected and, for each job i belonging to this set, a new branch is created where job i is constrained to be first in the sequence among the jobs in this set. Thus, at each node of the search tree a partial sequence σ associated with a partial nearly active schedule is considered. By *partial* solution, we mean that only a subset of jobs of N has been sequenced to be scheduled as soon as possible following the procedure described in the proof of Proposition 2.2. In fact, a partial solution σ represents the subset of schedules whose associated sequence starts with subsequence σ. Hereafter we denote as $m(\sigma)$ the earliest available machine in the partial nearly active schedule associated with σ. At each node of the search tree, an unsequenced job i is then sequenced at the end of σ corresponding to the schedule of i on machine $m(\sigma)$ as soon as possible. The sequence is built by using a depth-first strategy.

Throughout the search tree we dynamically maintain several sets of jobs that represent the current state of a partial solution σ:

- $F(\sigma)$ is the sum of the weighted tardiness of the jobs in the partial schedule associated with σ;
- $\overline{\sigma} = \{j \in N / i \notin \sigma\}$ is the set of jobs which are not sequenced in σ (that need to be sequenced after σ);
- $PF(\sigma) \subseteq \overline{\sigma}$ (possible first) is the set of jobs which it is possible to attempt to sequence first immediately after σ.

Some basic filtering methods are used in this enumeration procedure at each node of the search tree. First, since the subset of

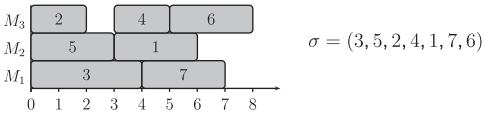

Fig. 1. A nearly active schedule and its associated sequence.

active schedules is dominant [5] for the problem, we eliminate from $PF(\sigma)$ any job that would give rise to a non-active schedule if it were to be scheduled first. Second, a simple rule based on the start times of jobs is also used (see for instance [33]): we only look at sequences that impose an order on scheduled jobs such that their start times are non-decreasing.

Other dominance rules are described in the following sections to filter the set $PF(\sigma)$ at each node of the search tree, thus enhancing the performance of the branch-and-bound procedure.

The following notation is also used in the remainder of the paper:

- $Start_i^\sigma$ is the start time of job i in the partial schedule associated with sequence σ;
- C_i^σ is the completion time of job i in the partial schedule associated with sequence σ;
- F_i^σ is the cost of job i in the partial schedule associated with sequence σ;
- A_j^σ is the availability time of machine j in the partial schedule associated with sequence σ;
- $C_i(t)$ is the completion time of job i if it is scheduled from date t *i.e.* $C_i(t)=\max(r_i,t)+p_i$;
- $F_i(t)$ is the cost of job i if it is scheduled from date t *i.e.* $F_i(t)=w_i \max(C_i(t)-d_i,0)$;
- $\sigma|\rho$ represents the sequence resulting from the concatenation of sequences σ and ρ.

Note that since a nearly active (partial) schedule S is associated with each (partial) sequence σ, the terms S and σ are used interchangeably below.

3. Machines on which a job can be scheduled

In this section we introduce the notion of machines on which a job can be scheduled (referred to as "possible machines") and we show how this notion can be used to improve the behavior of the enumeration method.

Given a partial solution σ and a job i, let μ_i^σ be the set of machines which can process job i in any (partial) schedule whose associated sequence begins with σ. At the root of the search tree, μ_i is set to M. If $i \in \sigma$ then μ_i^σ is a singleton containing only the machine m_i^σ which processes i in the schedule associated with σ. If $i \in \overline{\sigma}$ then $\mu_i^\sigma \subseteq M$. Note that the definition of μ_i implies that for two partial solutions σ and σ' such that σ' begins with σ, we have $\mu_i^{\sigma'} \subseteq \mu_i^\sigma$.

We now show how μ_i can be filtered throughout the search according to the jobs already scheduled. In [22], Jouglet et al. described a dominance rule on unscheduled jobs for the one-machine problems. Let σ be a partial solution. Let j and k be two jobs such that $j \in \overline{\sigma}$ and $k \in PF(\sigma)$. Jouglet et al. described conditions in which j dominates k in the first position. By "dominance in the first position", they mean that if k is scheduled immediately after sequence σ, and if the sequence $\sigma|k$ is completed by any sequence ρ of jobs belonging to $\overline{\sigma}/\{k\}$ including job j, then it will always be "better" to interchange j and k, *i.e.* the sequence $\sigma|k|\rho$ is dominated whatever sequence ρ is, and job k can be removed from the set $PF(\sigma)$ of possible first jobs. While this one-machine dominance rule cannot be used to filter the set $PF(\sigma)$ from a partial solution σ if $m>1$, it can, however, be used directly to filter the possible machines of unscheduled jobs in the following way. Let σ be a solution of a given parallel machines problem with $m>1$. Let job k be the last sequenced job in σ. Let j be a job such that $j \in \overline{\sigma}$. Suppose that before job k was processed, job j dominated job k in the first position, according to the single-machine dominance rule of [22]. We may deduce that from now on, if j is scheduled on the machine m_k^σ, then we will obtain a strictly better schedule by interchanging j and k or by inserting j before k. Therefore, we should not schedule job j on machine m_k^σ, and μ_j^σ may be set to $\mu_j^\sigma = \mu_j^\sigma \backslash \{m_k^\sigma\}$.

Note that this possibility of filtering the sets of possible machines is not exhaustive, and we believe that new deduction methods might be found that improve performance. The information can be used in different ways:

- If there exists an unscheduled job $i \in \overline{\sigma}$ such that $\mu_i^\sigma = \emptyset$ then the node associated with the partial schedule σ can be pruned from the search tree since any solution which can be derived from this node is strictly dominated.
- Suppose that $x \notin \mu_i^\sigma$ with $i \in \overline{\sigma}$. Then we also have $x \notin \mu_i^{\sigma|\rho}$, where ρ represents any sequence of a subset of $\overline{\sigma}/\{i\}$. Consequently, each time we encounter a partial solution where the next job to schedule is to be processed by machine x, *i.e.* a sequence $\sigma|\rho$ for which $m(\sigma|\rho)=x$, then job i can be removed from $PF(\sigma|\rho)$.
- The fact that some unscheduled jobs can be processed by only a subset of M can be used to improve some dominance rules. Such an improvement is proposed in Section 5.

4. Local dominance rule

In this section we show how we can extend to the parallel machine case the notion of a "LOWS-Active" schedule, used in solving one-machine problems.

Jouglet et al. [22,23] described a dominant subset of schedules termed LOWS-active (LOcally Well Sorted active) schedules. This is an improved generalization of a local dominance rule described by Chu [15,16] and Chu and Portmann [17] for non-weighted criteria, and it involves building schedules such that no exchanges of two adjacent jobs can improve the schedule.

The dominance rule described above may be applied simply to the parallel machine problem in a branch-and-bound procedure as follows:

Theorem 4.1 (*Jouglet et al. [22,23]*). *Consider a partial solution associated with sequence σ. Let j be the last job on the machine $m(\sigma)$ and let $k \in PF(\sigma)$. If $C_k(\Delta_j^\sigma) \le C_j^\sigma$ and $F_k(\Delta_j^\sigma)+F_j(C_k(\Delta_j^\sigma)) \le F_j^\sigma + F_k(C_j^\sigma)$ (with at least one strict inequality), then the schedule associated with sequence $\sigma|k$ is dominated and k is removed from $PF(\sigma)$, since it is better to schedule k before j.*

We now propose an adaptation of the LOWS-active property for the parallel machine case. We introduce the notion of "adjacent front", which allows us to define which jobs are the neighbors of a particular job. Then, we can check whether or not swapping two neighbor jobs is advantageous. Recall that m_i^σ is equal to the machine which processes i in the schedule associated with σ.

Definition 4.1. Consider a solution associated with sequence σ. Let $Pred_i^\sigma$ be the set of jobs preceding i in sequence σ. The adjacent front of i in σ, is defined by: $Adj_i^\sigma = \{j \in Pred_i^\sigma / \forall k \in Pred_i^\sigma, (m_j^\sigma = m_k^\sigma) \Rightarrow (Start_j^\sigma > Start_k^\sigma)\}$.

Informally speaking, Adj_i^σ is equal to the set of jobs scheduled in the final slot on each machine according to the partial schedule obtained just before scheduling i. Now we can introduce the dominant subset of the LOWS-active schedules for the parallel machine case.

Definition 4.2. An active schedule associated with sequence σ is said to be *LOWS*-active if

$\forall i \in \sigma$ and $\forall j \in Adj_i^\sigma$

1. if $m_j^\sigma \neq m_i^\sigma$, at least one of the following conditions holds:
   (a) $F_j^\sigma + F_i^\sigma < F_j(\Delta_i^\sigma) + F_i(\Delta_j^\sigma)$;

(b) $\min(C_j^\sigma, C_i^\sigma) < \min(C_j(\Delta_i^\sigma), C_i(\Delta_j^\sigma))$ or $\max(C_j^\sigma, C_i^\sigma) < \max(C_j(\Delta_i^\sigma), C_i(\Delta_j^\sigma))$;
(c) $F_j^\sigma + F_i^\sigma = F_j(\Delta_i^\sigma) + F_i(\Delta_j^\sigma)$ and $\min(C_j^\sigma, C_i^\sigma) = \min(C_j(\Delta_i^\sigma), C_j(\Delta_i^\sigma))$ and $\max(C_j^\sigma, C_i^\sigma) = \max(C_j(\Delta_i^\sigma), C_i(\Delta_j^\sigma))$.

2. if $m_j^\sigma = m_i^\sigma$, at least one of the following conditions holds:
(a) $F_j^\sigma + F_i^\sigma < F_i(\Delta_j^\sigma) + F_j(C_i(\Delta_j^\sigma))$;
(b) $C_i^\sigma < C_j^\sigma(C_i(\Delta_j^\sigma))$;
(c) $F_j^\sigma + F_i^\sigma = F_i(\Delta_j^\sigma) + F_j(C_i(\Delta_j^\sigma))$ and $C_i^\sigma = C_j(C_i(\Delta_j^\sigma))$.

A schedule associated with σ is said to be LOWS-active if none of the jobs i in the schedule can be exchanged with another job j in i's adjacent front so as to create a better schedule. Conditions are established so that any such exchange of jobs either increases the cost locally or delays jobs scheduled after j. This definition means that a schedule can be discarded where none of the conditions (a), (b) and (c) are met. In such schedules, exchanging jobs i and j leads to a schedule at least as good as the original one. Indeed, as conditions (a) are not fulfilled, the swap does not increase the sum of costs of jobs i and j. As conditions (b) do not hold either, jobs scheduled after i and j are not delayed by this interchange and their cost does not increase. As conditions (c) are also not fulfilled, it ensures that the interchange either strictly decreases the sum of costs of i and j or let the machines available strictly earlier after the schedule of i and j.

Note that the case $m_j^\sigma = m_i^\sigma$ corresponds exactly to the conditions established by Jouglet et al. [22,23] for obtaining a LOWS-active schedule on one machine. As for the one-machine problem, we can now propose the following theorem, proved by considering the different possible interchange cases:

Theorem 4.2. *The subset of the LOWS-active schedules is dominant for the parallel machine total weighted tardiness problem.*

Proof. We have to prove that there always exists an optimal schedule that is also *LOWS*-active. At first, it is well known that the set of active schedules is dominant, *i.e.* there is at least an optimal schedule which is active [5]. Consider an active schedule associated with sequence σ which is not *LOWS*-active. There exist two jobs i and j for which the condition of *LOWS*-active property is not met. First, suppose that i and j are scheduled on different machines. None of the corresponding conditions (a), (b) and (c) holds. This means that jobs i and j can be exchanged without delaying any other job and without increasing the cost (see above). Moreover, because condition (c) does not hold before the swap, we know that at least one of these conditions holds after swapping i and j. Now, if jobs i and j are scheduled on the same machine, in the same way the two jobs can be exchanged without delaying any other job and without increasing the cost [23]. This procedure can be applied on the schedule each time a couple of jobs i and j that do not have the *LOWS*-active property is found. During these swaps, the cost never increases. This means that the resulting schedule is both optimal and *LOWS*-active. □

The dominance rule is used in the branch-and-bound procedure as follows: if the schedule corresponding to $\sigma|k$ is not LOWS-active, then the sequence $\sigma|k$ is dominated and k is removed from $PF(\sigma)$. Note that the adjacent front of k for the test corresponds exactly to the set of jobs last scheduled on each machine. Since there are at most m jobs to consider in this adjacent front, testing if $\sigma|k$ is *LOWS*-active runs in $O(m)$ time.

5. Dominated sequences

In this section we extend the notion of "dominated sequences" defined for the one-machine problem. Then we show how this notion can be used within the enumeration procedure to reduce the search space. Here we shall focus on sequences, but recall that these sequences are associated with nearly active schedules (see Section 2) and that we might just as easily talk about schedules as sequences.

For some flow-shop scheduling problems, Ignall and Schrage [21] showed how to compare two nodes of the search tree which sequence the same set of jobs. In fact, this kind of dominance relation can be adapted to the $Pm|r_i|\sum w_iT_i$ problem.

Thus, it is possible to compare two partial sequences σ and σ' from the same set of jobs (*i.e.* σ' is a permutation of σ). Informally speaking, a sequence σ' dominates a sequence σ if σ may be replaced advantageously by σ' in any sequence starting with sequence σ. It is clearly a dominance relation between the set of solutions beginning with sequence σ and the set of solutions beginning with sequence σ'. Now, suppose that, when enumerating solutions, we encounter a partial solution σ. If we can find such a sequence σ' which dominates σ then the implicit enumeration of solutions derived from σ can be stopped if one of the two following conditions is satisfied (see for example [24]):

1. σ' dominates strictly σ and thus σ cannot lead to an optimal solution;
2. σ' is only at least as good as σ but we are sure that the best solution of σ' was or will be enumerated during the search.

Several methods exist for finding better permutations σ' of σ (see [22]). For this problem we use a set of sequences already encountered during the search as explained below. In the following, recall that $\overline{\sigma}$ is the set of jobs not belonging to sequence σ, *i.e.* $\overline{\sigma} = \{l \in N | l \notin \sigma\}$. Note that if σ' is a permutation of σ, we have $\overline{\sigma'} = \overline{\sigma}$.

Now, consider any schedule and a subset X of jobs in this schedule. Suppose that we delay the jobs belonging to X by δ time units. Let us define $\overline{\Lambda}_X(\delta)$ as any nondecreasing upper bound of the difference between the total cost of jobs belonging to X before and after delaying these jobs by δ time units, whatever their positions in the schedule. For the total weighted tardiness the evaluation function $\overline{\Lambda}_X(\delta) = \delta\sum_{i \in X}\{w_i\}$ is valid. This function considers the worst case in which all jobs are all already late before delaying them from δ. Thus, for each job $i \in X$, a cost δw_i is added.

For the special case with $m=1$, Jouglet et al. [22] identified the following conditions in which a partial sequence of jobs σ is dominated by another sequence σ' from the same set of jobs. If σ is a one-machine sequence, we denote by A^σ the availability time of the machine, *i.e.* the completion time of the last job in σ.

Proposition 5.1 (*Jouglet et al. [22]*). *Consider a partial schedule on a single machine associated with a sequence σ. Consider a sequence σ' such that σ' is a permutation of σ. If one of the following two conditions holds*:

1. $A^{\sigma'} \leq \max(A^\sigma, \min_{j \in \overline{\sigma}}\{r_j\})$ *and* $F(\sigma') \leq F(\sigma)$;
2. $F(\sigma') + \overline{\Lambda}_{\overline{\sigma}}(\delta) \leq F(\sigma)$, *where* $\delta = \max(0, A^{\sigma'} - \max(A^\sigma, \min_{j \in \overline{\sigma}}\{r_j\}))$

then σ is dominated.

Recall that $F(\sigma)$ is the total weighted tardiness of the jobs in the schedule associated with σ. The first condition concerns the case where no delay is entailed for the jobs in $\overline{\sigma}$ when σ is replaced by σ', and the second condition concerns the case where jobs in $\overline{\sigma}$ will be delayed by δ units of time. Such a delay occurs when, rescheduling jobs according to σ', new idle times are added. Thus, $\overline{\Lambda}_{\overline{\sigma}}(\delta)$ evaluates the maximum cost of this delay.

We shall now show how this dominance rule can be adapted and generalized to the parallel machine case.

First, since there is a subsequence of jobs associated with each machine in a schedule of the parallel machine problem, the previous one-machine dominance rule can be used. Consider a

partial solution associated with sequence σ. We now denote by σ_i the subsequence of jobs on machine i. If one of the subsequences σ_i is strictly dominated (in terms of cost) by a sequence σ' of the same set of jobs according to the previous dominance rule, then the whole partial solution cannot lead to an optimal solution. Note that for the second condition, we do not need to consider all the jobs belonging to $\overline{\sigma}$ in the evaluation of the additional cost due to the delay of certain jobs. Indeed, when considering subsequence σ^i, we can take the subset $\overline{\sigma}_i = \{j \in \overline{\sigma} / i \in \mu_j^\sigma\}$ instead of $\overline{\sigma}$ since only these jobs will possibly be scheduled after σ_i on machine i. This is one of the benefits of the notion of "possible machines" (see Section 3). Applying to the parallel machine problem, the dominance rule is then expressed as:

Proposition 5.2. *Consider a partial schedule on m machines associated with a sequence σ. If one of the subsequence $\sigma_i, i \in M$ is such that there exists a schedule on a single machine associated with one-machine sequence σ' of the set of jobs belonging to σ_i for which one of the following two conditions holds:*

1. $A^{\sigma'} \leq \max(A_i^\sigma, \min_{j \in \overline{\sigma}_i}\{r_j\})$ *and* $F(\sigma') < F(\sigma)$;
2. $F(\sigma') + \overline{\Lambda}_{j \in \overline{\sigma}_i}(\delta) < F(\sigma)$, *where* $\delta = \max(0, A^{\sigma'} - \max(A_i^\sigma, \min_{j \in \overline{\sigma}_i}\{r_j\}))$;

then σ cannot lead to an optimal solution.

Computing the sum of weights and the minimum release dates of jobs belonging to $\overline{\sigma}^i$ takes $O(|\overline{\sigma}|)$ time. Thus, given sequences σ_i and σ', it takes $O(|\overline{\sigma}|)$ time to know if σ_i is strictly dominated by σ', given that it has already been established that σ' is a permutation of σ_i and that the set of values $\{(F(\sigma_i), A_i^\sigma), (F(\sigma'), A^{\sigma'})\}$ is known.

We shall now extend the previous dominance rule by considering the whole sequence σ of jobs instead of each subsequence σ_i. The difficulty now comes from the fact that the completion time A_i^σ of each machine i has to be taken into account in the conditions of the dominance rule.

Given a sequence σ, we now define the following strict total order relation $\lhd_\sigma$ over the index of machines: $i \lhd_\sigma j$ if and only if $A_i^\sigma < A_j^\sigma$ or $(A_i^\sigma = A_j^\sigma$ and $i < j)$. Then, we denote by $[i]^\sigma$ the ith value in the sorted vector of the machine indexes according to $\lhd_\sigma$.

The extension of the dominance rule to the parallel machine case can then be expressed as:

Proposition 5.3. *Consider a partial schedule associated with sequence σ. Consider a sequence σ' such that σ' is a permutation of σ. If one of the following two conditions holds*:

1. $\forall i \in M$, $A_{[i]^{\sigma'}}^{\sigma'} \leq \max(\min_{j \in \overline{\sigma}_{[i]^\sigma}}\{r_j\}, A_{[i]^\sigma}^\sigma) \wedge F(\sigma') \leq F(\sigma)$
2. $F(\sigma') + \sum_{j \in \overline{\sigma}} \overline{\Lambda}_j(\delta_j) \leq F(\sigma)$,
   where $\delta_j = \max_{[i]^\sigma \in \mu_j^\sigma}\left(\max\left(0, A_{[i]^{\sigma'}}^{\sigma'} - \max\left(\min_{j \in \overline{\sigma}_{[i]^\sigma}}\{r_j\}, A_{[i]^\sigma}^\sigma\right)\right)\right)$

then σ is dominated.

Proof. Once again, the first condition concerns the case where, after reconfiguring, replacing σ by σ' entails no delay for the jobs in $\overline{\sigma}$, while the second condition is where some jobs in $\overline{\sigma}$ are delayed.

Consider two sequences σ and σ' of the same set of jobs with $F(\sigma') \leq F(\sigma)$. Let S be any schedule whose associated sequence starts with σ. Let τ_i be the sequence of jobs which completes sequence σ_i on machine i in S, *i.e.* the complete subsequence of jobs on machine i in S is $\sigma_i|\tau_i$. Note that each job j belonging to τ_i is such that $i \in \mu_j^\sigma$. Thus, in S sequence τ_i cannot start earlier than $t_i = \max(A_k^\sigma, \min_{j \in \overline{\sigma}^i}\{r_j\})$.

Let S' be the schedule obtained from S, σ and σ' in the following way. The first part of S' is built by scheduling jobs belonging to σ by following sequence σ' using the procedure described in the proof of Proposition 2.2. Then, on each machine $[i]^{\sigma'}$, jobs belonging to $\tau_{[i]^\sigma}$ are scheduled in S' as soon as possible from time $\max(t_{[i]^\sigma}, A_{[i]^{\sigma'}}^{\sigma'})$ following sequence $\tau_{[i]^\sigma}$. Note that each job $j \in \overline{\sigma}$ scheduled on machine $[i]^\sigma$ in S is delayed of at most $\max\{0, A_{[i]^{\sigma'}}^{\sigma'} - \max\{\min_{j \in \overline{\sigma}_{[i]^\sigma}}\{r_j\}, A_{[i]^\sigma}^\sigma\}\}$. Note also that we have $F(S) - F(S') = F(\sigma) - F(\sigma') + \sum_{j \in \overline{\sigma}}(F_j^S - F_j^{S'})$ and thus $F(S') = F(S) - F(\sigma) + F(\sigma') - \sum_{j \in \overline{\sigma}}(F_j^S - F_j^{S'})$.

First, suppose that condition (1) holds. Then $\forall i \in M$, we have $A_{[i]^{\sigma'}}^{\sigma'} \leq \max\{\min_{j \in \overline{\sigma}_{[i]^\sigma}}\{r_j\}, A_{[i]^\sigma}^\sigma\}$. Thus, each job $j \in \overline{\sigma}$ is scheduled at the same time in S and S' and we have $F_j^S = F_j^{S'}$. It follows that $F(S') \leq F(S)$.

Now, suppose that condition (2) holds. In S', a job $j \in \overline{\sigma}$ is processed by a machine $[i]^\sigma$ belonging to μ_j^σ. Thus, job j has been delayed by at most $\delta_j = \max_{[i]^\sigma \in \mu_j^\sigma}(\max(0, A_{[i]^{\sigma'}}^{\sigma'} - \max\{\min_{j \in \overline{\sigma}_{[i]^\sigma}}\{r_j\}, A_{[i]^\sigma}^\sigma\}))$ and $F_j^{S'} \leq F_j^S + \overline{\Lambda}_j(\delta_j)$. It follows that $F(\sigma) - F(\sigma') - \sum_{j \in \overline{\sigma}}(F_j^S - F_j^{S'}) \geq 0$ and $F('S) \leq F(S)$. □

During the construction of S' the subsequence $\sigma'_{[i]}$ could have been completed with a subsequence other than $\tau_{[i]}$. However, the chosen matching is the one that minimizes the longest possible delay among jobs. In this evaluation, it could have also been assumed that in the worst case the cost of all jobs belonging to $\overline{\sigma}$ will be increased, due to the longest delay over the jobs which can appear on one of the machines if a schedule beginning by σ is replaced by σ'. Nevertheless, like for the single-machine dominance rule applied to the parallel machine case, we make use of the notion of possible machines to improve this evaluation, by considering the longest possible delay δ_j for each job j considering the set of machines which can potentially process it. Given two sequences σ and σ', it takes $O(m(\log m + |\overline{\sigma}|)$ time to know if σ is dominated by σ', by considering we already know that σ' is a permutation of σ and the set of values $\{(F(\sigma), A_{[1]^\sigma}^\sigma, \ldots, A_{[m]^\sigma}^\sigma), (F(\sigma'), A_{[1]^{\sigma'}}^{\sigma'}, \ldots, A_{[m]^{\sigma'}}^{\sigma'})\}$.

We remark that Proposition 5.3 may be used with only a subset of the m machines. In fact, in an m-machine schedule, if a subsequence associated with a schedule considering only k machines over m is dominated, it is obvious that the entire schedule is dominated. However, it is not easy to test this rule in practice, because every possible subset of machines needs to be investigated. After some fruitless attempts, we abandoned this possibility.

To apply these dominance rules to a current partial solution σ associated with a node of the search tree, we need to find sequences (or subsequences) σ' which allow us to establish that σ is dominated. Of course, enumerating all such possible sequences is not feasible in practice. During the search with our enumerative method, we only consider sequences recorded from previously encountered solutions. Using suitable data structures, relevant recorded sequences can be found in an effective way.

Consider a partial solution σ associated with a particular node of the search tree. As soon as all solutions which can be derived from this node have been (implicitly) enumerated, *i.e.* as soon as the best solution which can be built from σ is known, the characteristic of σ are stored in two sets V_1 and V_m. To this end, only the following data are saved:

- the set of jobs belonging to sequence σ associated with the pair of values $(F(\sigma), A_{[1]^\sigma}^\sigma, \ldots, A_{[m]^\sigma}^\sigma)$ is recorded in V_m;
- for each machine i, the set of jobs belonging to sequence σ_i associated with the pair of values $(F(\sigma_i), A_i^\sigma)$ is recorded in V_1.

During the search, the previously recorded solutions are used in the following way at each node of the search tree. Let σ be the partial solution associated with the examined node. The set of possible first jobs $PF(\sigma)$ is filtered. For all $j \in PF(\sigma)$, we seek a state in V_m whose characteristics meet the condition of Proposition 5.3 to deduce that sequence $\sigma|j$ is dominated. If such a state is found then scheduling a job j immediately after σ yields a partial solution

which is dominated, *i.e.* no solution better than those previously encountered can be built from $\sigma|j$. Thus, job j can be removed from set $PF(\sigma)$. If it is not the case, job j could be sequenced immediately after σ, *i.e.* scheduled as soon as possible on machine $m(\sigma)$. The subsequence $\sigma_{m(\sigma)}|j$ is then tested by searching a state in V_1 whose characteristics meet the condition of Proposition 5.2. If it is the case, job j can also be removed from set $PF(\sigma)$. Note that it is required in Proposition 5.2 that the state obtained from V_1 is strictly better than $\sigma_{m(\sigma)}|j$ (according to the cost) to eliminate j from set $PF(\sigma)$. An equivalence is not sufficient to deduce that no better solution than previously encountered ones can be built from $\sigma|j$, given that subsequences other than $\sigma_{m(\sigma)}$ are not considered.

Hash tables combined with height-balanced binary search trees make for an efficient use of sets V_1 and V_m. This generally allows us to obtain a pertinent state of one of the sets in $O(n \log n)$. A state of a set consists of a list of jobs sorted in lexicographic order coded in a vector, and a list of couples (F, A) corresponding to the different states of the nodes which have been visited within this set of jobs. Note that only non-comparable couples can be retained. Suppose that we search a state with the same set of jobs belonging to a sequence σ in one of set V_1 or V_m. Sorting jobs in lexicographic order runs in $O(n \log n)$ time. Computing an index in the hash table of this set of jobs runs in $O(n)$ time. Since jobs are sorted in lexicographic order, a comparison of two sets of jobs runs in $O(n)$ time. Thus, if there is no collision with another state at the same index in the hash table, seeking the set of relevant sequences σ' encountered from previously encountered solution in the search runs in $O(n \log n)$ time. In case of collision with k states at the same index in the hash table, a height-balanced binary search tree is used with an additional search running in $O(n \log k)$ time. Once a relevant state has been found with the same set of jobs, it simply needs to be determined whether a couple (F,A) of the state satisfies the sufficient condition of Proposition 5.3 or 5.2 according to the case. Experimental tests revealed that the amount of such (non-comparable) couples is very low.

6. Experimental results

In this section we provide experimental results to compare the effectiveness of the described dominance rules. All the computations were performed on a 1.6 GHz Pentium-M running MS-Windows XP. Instances were generated from schemes used in the literature.

We used a classical branch-and-bound procedure given in Algorithm 1. A depth first strategy is used within this enumeration method using a stack *STACK* of nodes. In the search tree, a node corresponds to a partial sequence σ. *STACK* is initialized with the empty sequence. A variable *UB* is used to store the value of the best found sequence *BestSequence*. A lower bound function *LB* (see below) is used to evaluate the minimal cost of a given solution. Dominance rules are applied to each generated node σ, leading possibly to eliminating some children of this node in the tree by filtering $PF(\sigma)$. When all the possible children of σ have been enumerated, the sequence σ is recorded in V_1 and V_m as explained in Section 5.

Algorithm 1. The branch-and-bound algorithm

```
Push the empty sequence on STACK;
V1 ← ∅, Vm ← ∅, UB ← ∞;
while STACK is not empty do
    Let σ be the sequence on the top of STACK;
    if PF(σ) ≠ ∅ then
        Let j ∈ PF(σ);
        PF(σ) ← PF(σ)/{j};
        σ' ← σ|j;
        if LB(σ') < UB then
            Filter PF(σ') according to Definition 4.2 as explained
            in Section 4;
            Filter PF(σ') according to Propositions 5.3and 5.2
            as explained in Section 5;
            Push σ' in STACK;
    else
        Remove the sequence σ which is on the top of STACK;
        Add σ to Vm and each subsequence σi, i ∈ M to V1;
        if all jobs belonging to N belong to σ and F(σ) < UB then
            UB ← F(σ);
            BestSequence ← σ;
```

To generate instances, we adapted the schemes from Chu [15] and Akturk and Ozdemir [1] for the single machine problem. For each n, m, we generate m sets of $\lfloor n/m \rfloor$ jobs and one set of $n-m*\lfloor n/m \rfloor$ jobs. This method is controlled by two parameters α and β. Values r_i, p_i, d_i and w_i are generated from uniform distributions: p_i are uniformly distributed in [1, 100], w_i in [1, 10]. Then, r_i are distributed in $[0,\alpha \sum p_i]$ and $d_i - r_i + p_i$ in $[0,\beta \sum p_i]$. Parameter α takes values {0,0.5,1,1.5} and β takes values in {0.05,0.25,0.5}. When n, m, α and β are fixed, 10 instances are created. Finally, there are 120 instances for each couple (n,m).

Tests were done on instances of different sizes in terms of the number of jobs. For each size there were three sets of instances, depending on the number of machines: 2, 3 or 5. A time limit of 1800 s was fixed.

In Table 1 we compare the effectiveness of the different techniques described in this paper. Below, "*LOWS*" refers to the use of the technique based on local dominance rule described in Section 4 while "DR_1" and "DR_m" respectively mean the use of techniques based on Propositions 5.2 and 5.3 described in Section 5. The enumeration procedure is run with instances of $n=15$ jobs and $m=\{2,3,5\}$ machines. For these tests we use a trivial lower bound at each node of the search tree. For each partial schedule, the lower bound is obtained by computing the total cost, assuming that all jobs will be scheduled as early as possible at the end of the partial

Table 1
Testing the effectiveness of the techniques, $n=15$.

m	%	*Basic*			*LOWS*		
		cpu	*nodes*	*opt.*	*cpu*	*nodes*	*opt*
2	81.67	27523	229133	95	1931	23945	100
3	74.17	10874	85484	86	2562	28074	100
5	75.00	132	1369	79	105	1163	83
m	%	DR_1			DR_m		
		cpu	*nodes*	*opt.*	*cpu*	*nodes*	*opt*
2	81.67	7906	26780	98	3112	8135	100
3	74.17	20478	34175	83	3939	9517	100
5	75.00	304	1268	75	267	900	100
m	%	*LOWS*, DR_1, DR_m			*LOWS*, DR_m		
		cpu	*nodes*	*opt.*	*cpu*	*nodes*	*opt*
2	81.67	711	1878	100	574	1961	100
3	74.17	1013	2237	100	744	2272	100
5	75.00	177	426	100	138	427	100

schedule. The method only uses the dominance rule of the active schedule and the start-times-based dominance rule (see Section 2). The results are reported in the column "*basic*". Techniques are first tested one by one ("*LOWS*", "DR_1", "DR_m"). Then, we report the results obtained when all techniques are used ("*LOWS*, DR_1, DR_m"). Finally, we present the results for the configuration of techniques that appears to be the best tradeoff between the size of the search space and the computation time efficiency, *i.e.* "*LOWS*, DR_m". Note that for the two last cases, we also used another dominance rule based on not scheduled jobs (see [29]) whose contribution adds virtually nothing to the results.

For the different numbers of machines we report the average computing times in milliseconds ("*cpu*") and the average number of generated nodes ("*nodes*") over the 120 generated instances. Note that some instances are not solved within the time limit. This is why we provide these statistics only over the instances which have been solved for all versions. The percentage of concerned instances is shown in the "%" column. However, we provide the percentage of instances which were solved for each version (columns "*opt.*").

We can see that all the rules are effective in reducing the search space, and especially the rule DR_m, which succeeds in reducing it quite dramatically. Nevertheless, the rate of improvement decreases as the number of machines grows. We can see that although DR_1 reduces the search space, computing times are higher than for the basic version where $m=3$. In the light of our numerical experiments we decided not to use DR_1 when the other dominance rules were activated. In fact, the size of the search space was very similar with or without DR_1, but the computation time was substantially higher.

Results of the branch-and-bound method using the best combination of the techniques (*LOWS*, DR_m) are shown in Table 2. The lower bound $lb_{[]combo}$ of Baptiste et al. [8] is used.

For the different combinations of parameters (α,β) and for different sizes of instances the average computing times in milliseconds (*cpu*) are shown, as well as the average number of generated nodes (*nodes*) over the 10 generated instances. Note that some instances are not solved within the time limit, and in these cases we provide in brackets the number of instances over 10 which are not solved. The *cpu* and *nodes* statistics are then provided only for the solved instances. As we can see, the hardest instances to solve are the ones with all release dates equal to 0 ($\alpha=0$). This suggests that a specific method for the instances without release dates should be used. In the case of different release dates ($\alpha>0$) the hardest instances are those with ($\alpha=0.5,\beta=0.25$) or ($\alpha=0.5,\beta=0.5$), as in the case of the $1|r_i|\sum w_iT_i$ problem [22]. Above $n=20$ jobs the instances become very hard to solve by this branch-and-bound method if the number of machines is greater than 3. This is hardly surprising, since to our knowledge even the most effective methods for the one-machine problem are able to solve all instances only as long as the number of jobs is less than 30 [22]. In both the one-machine and parallel machine cases, this can be explained by the fact that the best-known lower bounds for these problems are not particularly satisfactory. Both the one-machine

Table 2
Computational results.

n	α	β	$m=2$		$m=3$		$m=5$	
			cpu	*nodes*	*cpu*	*nodes*	*cpu*	*nodes*
10	0	0.05	105	360	286	908	394	922
	0	0.25	120	400	253	801	337	783
	0	0.5	109	380	169	540	122	248
	0.5	0.05	25	79	36	121	22	91
	0.5	0.25	28	113	28	100	33	120
	0.5	0.5	41	140	42	156	8	28
	1	0.05	12	31	10	35	9	33
	1	0.25	6	24	16	47	3	14
	1	0.5	8	25	6	19	3	6
	1.5	0.05	6	21	5	25	6	19
	1.5	0.25	6	9	5	15	2	0
	1.5	0.5	2	6	3	13	2	1
15	0	0.05	2273	4659	13 884	26 868	79 984	78 104
	0	0.25	2924	5614	25 551	47 681	228 389	166 394
	0	0.5	4608	9532	72 542	118 595	155 381	85 746
	0.5	0.05	1264	2614	1477	3180	702	1483
	0.5	0.25	1406	3158	5955	9921	3964	6853
	0.5	0.5	2044	4842	1388	3308	134	324
	1	0.05	117	293	109	267	88	207
	1	0.25	56	142	31	78	37	70
	1	0.5	69	155	39	90	8	9
	1.5	0.05	42	90	205	453	5	12
	1.5	0.25	9	29	11	27	6	2
	1.5	0.5	6	12	3	3	5	0
20	0	0.05	30 106	40 407	473 142	489 333	1 494 446 (**10**)	913 122
	0	0.25	78 533	89 729	584 889 (**2**)	569 814	–	_
	0	0.5	245 462	337 232	1 224 438 (**3**)	1 082 238	1 730 781 (**9**)	576 345
	0.5	0.05	50 830	71 230	117 463	153 602	137 261	109 252
	0.5	0.25	66 991	92 108	311 556	358 236	163 428 (**2**)	124 420
	0.5	0.5	42 016	62 340	199 344 (**1**)	221 334	85 912 (**2**)	62 731
	1	0.05	881	1611	4334	6698	12 626	10 738
	1	0.25	1425	2698	3227	4936	106	176
	1	0.5	25	41	594	974	192	334
	1.5	0.05	1176	1776	53	104	30	44
	1.5	0.25	3	9	34	64	60	105
	1.5	0.5	2	0	6	5	13	16

and the parallel machine total weighted tardiness problem appear very hard to solve in practice.

7. Conclusion

This paper describes dominance rules for the parallel total weighted tardiness scheduling problem. We have shown how it is possible to deduce whether or not some jobs can be processed by a particular machine in a particular context. This idea is used in non-trivial adaptations of one-machine dominance rules from the literature. A numerical validation of the described technique is presented. Experimental results show that the dominance rules are effective in reducing the search space and the computing times of a branch-and-bound method. The absence of a satisfactory lower bound for this problem means that if the number of machines is too high, then it is possible to solve all instances only as long as the number of jobs does not exceed $n=20$. Nevertheless, considering that the best methods for the one-machine version of the problem are only able (to our knowledge) to solve instances up to $n=30$ jobs, and considering that the hardness of the problems dramatically increases with the number of machines, the obtained results would appear to be reasonably good. Moreover, note that without the use of the dominance rules described in this paper, a lot of instances of 15 jobs cannot be solved, which demonstrates the interest of the described techniques. To our knowledge, ours is the first attempt to solve this problem exactly.

Acknowledgments

The authors would like to thank Jacques Carlier for enlightening discussions on this problem. They also would like to thank the reviewers, whose remarks helped to improve this paper.

References

[1] Akturk MS, Ozdemir D. An exact approach to minimizing total weighted tardiness with release dates. IIE Transactions 2000;32:1091–101.
[2] Akturk MS, Ozdemir D. A new dominance rule to minimize total weighted tardiness with unequal release dates. European Journal of Operational Research 2001;135:394–412.
[3] Azizoglu M, Kirca O. Tardiness minimization on parallel machines. International Journal of Production Economics 1998;55:163–8.
[5] Baker KR. Introduction to sequencing and scheduling. John Wiley and Sons; 1974.
[6] Baptiste Ph. Scheduling equal-length jobs on identical parallel machines. Discrete Applied Mathematics 2000;13:21–32.
[7] Baptiste Ph, Carlier J, Jouglet A. A branch-and-bound procedure to minimize total tardiness on one machine with arbitrary release dates. European Journal of Operational Research 2004;158(3):595–608.
[8] Baptiste Ph, Jouglet A, Savourey D. Lower bounds for parallel machine scheduling problems. International Journal of Operational Research 2008;3(6): 661–82.
[12] Brucker P. Scheduling algorithms. Springer Lehrbuch; 1995.
[13] Carlier J. Ordonnancements à contraintes disjonctives. RAIRO 1978;12: 333–351.
[15] Chu C. A branch and bound algorithm to minimize total flow time with unequal release dates. Naval Research Logistics 1992;39:859–75.
[16] Chu C. A branch and bound algorithm to minimize total tardiness with different release dates. Naval Research Logistics 1992;39:265–83.
[17] Chu C, Portmann M-C. Some new efficients methods to solve the $n|1|r_i|\sum T_i$. European Journal of Operational Research 1991;58:404–13.
[18] Conway RW, Maxwell WL, Miller LW. Theory of scheduling. Reading: Addison-Wesley; 1967.
[21] Ignall E, Schrage L. Applications of the branch-and-bound technique to some flow-shop scheduling problems. Operations Research 1965;13:400–12.
[22] Jouglet A, Baptiste Ph, Carlier J. Branch-and-bound algorithms for total weighted tardiness. In: Leung JY-T, editor. Handbook of scheduling: algorithms models and performance analysis, vol. 13. Chapman & Hall/CRC edition; 2004. p. 1–21.
[23] Jouglet A, Savourey D, Carlier J, Baptiste Ph. Dominance-based heuristics for one-machine total cost scheduling problems. European Journal of Operational Research 2008;184(3):879–99.
[24] Kohler WH, Steiglitz K. Enumerative and iterative computational approaches. In: Coffman EG, editor. Computer and job-shop scheduling theory, vol. 6. John Wiley & Sons; 1976. p. 229–87.
[25] Lenstra J, Kan AR, Brucker P. Complexity of machine scheduling problems. Annals of Discrete Mathematics 1977;1:343–62.
[26] Liaw C-F, Lin Y-K, Cheng C-Y, Chen M. Scheduling unrelated parallel machines to minimize total weighted tardiness. Computers & Operations Research 2003:1777–89.
[27] Mitten L. Branch-and-bound methods: general formulation and properties. Operations Research 1970;18(1):24–34.
[28] Nessah R, Yalaoui F, Chu C. A branch-and-bound algorithm to minimize total weighted completion time on identical parallel machines with job release dates. Computers & Operations Research 2008;35(4):1176–90.
[29] Savourey D. Ordonnancement sur machines parallèles: minimiser la somme des coûts. PhD thesis, Université de Technologie de Compiègne, France; 2006.
[30] Shim S-O, Kim Y-D. Scheduling on parallel identical machines to minimize total tardiness. European Journal of Operational Research 2007:135–46.
[31] Walker R. An enumerative technique for a class of combinatorial problems. American Mathematical Society Symposia in Applied Mathematics 1960;10: 91–4.
[32] Yalaoui F, Chu C. Parallel machine scheduling to minimize total tardiness. International Journal of Production Economics 2002;76:265–79.
[33] Yalaoui F, Chu C. New exact method to solve the $Pm|r_i|\sum C_i$ problem. International Journal of Production Economics 2006;100(1):168–79.