# Dominance-Based Heuristics for One-Machine Total Cost Scheduling Problems

Antoine Jouglet, David Savourey, Jacques Carlier, Philippe Baptiste

## HAL Id: hal-00155576
## https://hal.science/hal-00155576

Submitted on 18 Jun 2007

# Dominance-Based Heuristics for One-Machine Total Cost Scheduling Problems

**Antoine Jouglet[1], David Savourey[1], Jacques Carlier[1] and Philippe Baptiste[2]**

[1] Heudiasyc, UMR CNRS 6599, Université de Technologie de Compiègne

BP 20 529, 60205 Compiègne, France

{antoine.jouglet, david.savourey, jacques.carlier}@hds.utc.fr

[2] Ecole Polytechnique, CNRS LIX, 91128 Palaiseau, France

philippe.baptiste@polytechnique.fr

## Abstract

We study the one-machine scheduling problem with release dates and we look at several objective functions including total (weighted) tardiness and total (weighted) completion time. We describe dominance rules for these criteria, as well as techniques for using these dominance rules to build heuristic solutions. We use them to improve certain well-known greedy heuristic algorithms from the literature. Finally, we introduce a Tabu Search method with a neighborhood based on our dominance rules. Experiments show the effectiveness of our techniques in obtaining very good solutions for all studied criteria.

**Key words:** Scheduling, Heuristics, Single Machine

## 1  Introduction

In this paper we consider the situation where $n$ jobs $J_1, \ldots, J_n$ have to be scheduled for processing on a single machine and where the objective is to minimize total (weighted) tardiness or total (weighted) completion time. Associated with each job $J_i$ 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, preemption is not allowed, and at most one job at a time can be scheduled on the machine. The tardiness of job $J_i$ is defined as $T_i = \max\left(\{0, C_i - d_i\}\right)$, where $C_i$ is the completion time of $J_i$. The problem is to find a feasible schedule with minimum total (weighted) tardiness $\sum (w_i)T_i$ or with minimum total (weighted) completion time $\sum (w_i)C_i$. These problems are denoted as $1|r_i| \sum w_i T_i$, $1|r_i| \sum T_i$, $1|r_i| \sum w_i C_i$ and $1|r_i| \sum C_i$.

Several Branch and Bound procedures have been described in the literature to solve these problems [1, 4, 5, 8, 9, 18]. To our knowledge, the best results are obtained for all the above mentionned criteria by Baptiste, Carlier and Jouglet [4, 18]. All these problems are

known to be NP-hard in the strong sense [21]. It is therefore preferable to use heuristics for large-scale problems, given that heuristics provide a reasonably good schedule with reasonable computing effort. Moreover, heuristics provide good upper bounds which can be used in Branch and Bound procedures to reduce the search tree. For total tardiness and total completion time, Chu and Portmann have described several heuristics which rely on local dominance properties [11, 10]. For total weighted tardiness, Akturk and Ozdemir [2] have also described a sufficient condition for local optimality which can improve heuristics. Some other heuristics from the literature can easily be adapted to the studied problems [19]. A large number of these heuristics are greedy algorithms: at each iteration, an unscheduled job is chosen in conformity with some priority rule (see for instance [11, 10, 23, 20]) and is added to a partial sequence of scheduled jobs. The heuristic stops when all jobs are scheduled. Other more sophisticated polynomial methods have been proposed [6, 7, 14]. Among them, the powerful Recovering Beam Search procedure proposed by Della Croce and T'Kindt [14] provides excellent results for the total completion time criterion.

In this paper we describe several heuristic methods relying on the use of dominance rules and other techniques. In Section 2, we describe some original dominance rules which are used in our methods. In Section 3, we recall some well known heuristics from the literature, and we propose two algorithms based on our dominance rules, which allow us to improve drastically the results of these heuristics. Moreover, we propose a new priority rule "CPRTWT" (Combined Priority Rule for Total Weighted Tardiness), based on local dominance properties, which can be used in the heuristics referred to above. Next, in Section 4 we describe a Tabu Search method with a new neighborhood which makes use of our dominance rules. We also describe some other efficient techniques, which allow us to improve the behavior of the Tabu Search method. Finally, we report some experimental results (Section 5) which show the effectiveness of all our methods in obtaining very good solutions to all the studied problems.

Hereafter, in order to simplify the presentation, we focus solely on the total weighted tardiness criterion. Note that all our results are also valid for all other criteria.

# 2   Dominance Rules

In this section we describe some dominance rules to be used in subsequent sections in order to improve the solutions built by heuristic methods.

## 2.1   Local Dominance Properties

We first describe local dominance properties and we define dominant subsets of schedules.

In an *active* schedule, no job can be completed earlier without delaying another job. An objective function which is non-decreasing with respect to all completion times is *regular*. Active schedules are dominant for regular criteria [3, 13].

Following Chu and Portmann [11, 10], we provide a necessary and sufficient condition

2

for local optimality. By "local optimality" Chu means the optimality of the sequence of two adjacent jobs in a given schedule (taking into account neither following nor preceding jobs).

Suppose that we have to schedule two jobs $J_j$ and $J_k$ on a machine available at time $t$. Let $WT_{jk}(t)$ be the sum of the weighted tardiness of jobs $J_j$ and $J_k$ obtained by scheduling $J_j$ before $J_k$ at time $t$, *i.e.*, $WT_{jk}(t) = w_j \max\{0, \max\{r_j, t\}\} + p_j - d_j\} + w_k \max\{0, \max\{\max\{r_j, t\} + p_j, \max\{r_k, t\}\} + p_k - d_k\}$.

**Proposition 1.** *Consider two jobs $J_j$ and $J_k$ which have to be scheduled on a machine available at time $t$. Scheduling $J_j$ before $J_k$ is locally optimal if and only if $WT_{jk}(t) - WT_{kj}(t) \le 0$.*

Relying on this condition, we then define a dominant subset of schedules. Let $\Delta_j(S)$ be the completion time of the job immediately preceding job $J_j$ in a schedule $S$. If $J_j$ is the first job of the schedule $S$, then $\Delta_j(S) = -\infty$.

**Definition 1.** *An active schedule $S$ is said to be **LO-Active** (a Locally Optimal Active Schedule) if and only if for any couple of consecutive jobs $J_j$ and $J_k$ one of the following conditions is met*

- *(1)* $\max\{r_j, \Delta_j(S)\} < \max\{r_k, \Delta_j(S)\}$,

- *(2)* $WT_{jk}(\Delta_j(S)) - WT_{kj}(\Delta_j(S)) \le 0$.

According to the previous definition, Proposition 1 leads to the following proposition.

**Proposition 2.** *All optimal active schedules of the total weighted tardiness one-machine problem are LO-Active.*

*Proof.* Assume that there exists an optimal active schedule $S$ which is not *LO*-Active. There is at least one pair of adjacent jobs $J_j$ and $J_k$ ($J_j$ followed by $J_k$) such that $WT_{jk}(\Delta_j(S)) - WT_{kj}(\Delta_j(S)) > 0$ and $\max\{r_j, \Delta_j\} \ge \max\{r_k, \Delta_j\}$. We construct another schedule $S'$ by interchanging $J_j$ and $J_k$ without moving any other job. This interchange does not delay jobs after $J_k$ since $\max\{r_k, \Delta_j\} \le \max\{r_j, \Delta_j\}$. Only the tardiness of jobs $J_j$ and $J_k$ are changed. It is then clear that $WT(S) - WT(S') = WT_{jk}(t) - WT_{kj}(t) > 0$. Hence, interchanging jobs $J_j$ and $J_k$ decreases strictly the total weighted tardiness and then $S'$ is strictly better than $S$. This contradicts the assumption that $S$ is an optimal schedule. $\square$

Not all *LO*-Active schedules are optimal and it is possible to remove some schedules which are dominated from the set of *LO*-Active schedules. We now remove schedules $S$ in which there are useless idle time periods, *i.e.*, in which there are two consecutive jobs $J_j$ and $J_k$ such that $WT_{jk}(\Delta_j(S)) - WT_{kj}(\Delta_j(S)) = 0$ and $\max\{r_j, \Delta_j(S)\} > \max\{r_k, \Delta_j(S)\}$.

**Definition 2.** *An active schedule $S$ is said to be **LOWS-Active** (Locally Optimal Well Sorted Active Schedule) if and only if for any couple of consecutive jobs $J_j$ and $J_k$ one of the following conditions is met*

3

- *(1)* $WT_{jk}(\Delta_j(S)) - WT_{kj}(\Delta_j(S)) < 0$,

- *(2)* $WT_{jk}(\Delta_j(S)) - WT_{kj}(\Delta_j(S)) = 0$ *and* $\max\{r_j, \Delta_j(S)\} \leq \max\{r_k, \Delta_j(S)\}$,

- *(3)* $WT_{jk}(\Delta_j(S)) - WT_{kj}(\Delta_j(S)) > 0$ *and* $\max\{r_j, \Delta_j(S)\} < \max\{r_k, \Delta_j(S)\}$.

According to the previous definition, we establish the following proposition.

**Proposition 3.** *The subset of the LOWS-Active schedules is dominant for the one machine total weighted tardiness problem.*

*Proof.* Consider an optimal schedule $S$ which is not *LOWS*-Active. There is at least one pair of adjacent jobs $J_j$ and $J_k$ ($J_j$ followed by $J_k$) for which none of the three conditions of a *LOWS*-Active schedule is satisfied. Condition (1) implies that $WT_{jk}(\Delta_j(S)) - WT_{kj}(\Delta_j(S)) \geq 0$, otherwise schedule $S$ should be *LOWS*-Active.
First, assume that $WT_{jk}(\Delta_j(S)) - WT_{kj}(\Delta_j(S)) = 0$. From condition (2), it follows that we have $\max\{r_j, \Delta_j(S)\} > \max\{r_k, \Delta_j(S)\}$. Now assume that $WT_{jk}(\Delta_j(S)) - WT_{kj}(\Delta_j(S)) > 0$. From condition (3), it follows that we have $\max\{r_j, \Delta_j(S)\} \geq \max\{r_k, \Delta_j(S)\}$. Note that in the two cases $\max\{r_j, \Delta_j(S)\} \geq \max\{r_k, \Delta_j(S)\}$.

We construct another schedule $S'$ by interchanging $J_j$ and $J_k$ without moving any other job. This interchange does not delay the jobs after $J_k$ since we have $\max\{r_k, \Delta_j\} \leq \max\{r_j, \Delta_j\}$. Only the tardiness of jobs $J_j$ and $J_k$ are changed. It is therefore clear that $WT(S) - WT(S') = WT_{jk}(t) - WT_{kj}(t) \geq 0$. Hence, interchanging jobs $J_j$ and $J_k$ does not increase total weighted tardiness. We can iterate this process until we obtain a *LOWS*-Active schedule. □

Note that this dominance rule dominates the dominance properties described by Akturk and Ozdemir [2], and Chu [11, 10].

## 2.2 "Better" Sequence

In this section we introduce the notion of "better" sequence, which allows us to compare two partial sequences $\sigma_1$ and $\sigma_2$ of the same set of jobs (*i.e.*, $\sigma_1$ is a permutation of $\sigma_2$). Informally speaking, we say that a sequence $\sigma_1$ is "better" than a sequence $\sigma_2$ if $\sigma_2$ can be replaced advantageously by $\sigma_1$ in any feasible schedule which starts with the sequence $\sigma_2$. $\sigma_2$ is then dominated by $\sigma_1$. Let $OJ$ be the set of jobs which do not belong to $\sigma_1$. Note that we have $OJ = \{J_l \in N | J_l \notin \sigma_1\} = \{J_l \in N | J_l \notin \sigma_2\}$. Consider any feasible schedule $S$ starting with sequence $\sigma_2$. Now, let us examine under what conditions sequence $\sigma_1$ is "as good as" sequence $\sigma_2$, *i.e.*, under what conditions it is possible to replace sequence $\sigma_2$ by sequence $\sigma_1$ in any feasible schedule $S$.

- If $C_{\max}(\sigma_1) \leq C_{\max}(\sigma_2)$ and $WT(\sigma_1) \leq WT(\sigma_2)$, then we can replace $\sigma_2$ by $\sigma_1$ in any feasible schedule $S$ starting with the sequence $\sigma_2$. Consequently, sequence $\sigma_1$ is at least "as good as" sequence $\sigma_2$.

4

- Now, assume that $C_{\max}(\sigma_1) > C_{\max}(\sigma_2)$. Let $r_{min} = \min_{\{J_l \in OJ\}}\{r_l\}$ be the smallest release date of jobs belonging to set $OJ$. If we replace $\sigma_2$ by $\sigma_1$ in a feasible schedule, all jobs in $OJ$ have to be shifted by at most $\max\{C_{\max}(\sigma_1), r_{min}\} - \max\{C_{\max}(\sigma_2), r_{min}\}$ time units. So, the additional cost for jobs in $OJ$ is at most $(\max\{C_{\max}(\sigma_1), r_{min}\} - \max\{C_{\max}(\sigma_2), r_{min}\}) \sum_{\{J_l \in OJ\}} w_l$. Hence, sequence $\sigma_1$ is at least "as good as" sequence $\sigma_2$ if we have $WT(\sigma_1) + (\max\{C_{\max}(\sigma_1), r_{min}\} - \max\{C_{\max}(\sigma_2), r_{min}\}) \sum_{\{J_l \in OJ\}} w_l \leq WT(\sigma_2)$.

Note that if $C_{\max}(\sigma_1) = C_{\max}(\sigma_2)$ and $WT(\sigma_1) = WT(\sigma_2)$, then $\sigma_1$ is equivalent to $\sigma_2$, and we can break ties by following the lexicographic order with respect to vectors of job indices. We can now define the notion of "better" sequence:

**Definition 3.** *A sequence $\sigma_1$ is "better" than a sequence $\sigma_2$ if $\sigma_1$ is at least "as good as" $\sigma_2$ and if either (1) $\sigma_2$ is not at least "as good as" $\sigma_1$ or (2) if $\sigma_1$ is lexicographically smaller than $\sigma_2$.*

We have shown in [18] that this dominance rule is very effective in improving Branch-and-Bound algorithms. The second case of the dominance rule does not appear very often in comparison with the first case. For total (weighted) tardiness, the considered upper bound of the additional cost for the delayed jobs of $OJ$ (*i.e.*, $(\max\{C_{\max}(\sigma_1), r_{min}\} - \max\{C_{\max}(\sigma_2), r_{min}\}) \sum_{\{J_l \in OJ\}} w_l$) is often loose. Nevertheless, the second case appears much more often for total (weighted) completion time, where this upper bound is tight.

# 3 Greedy Heuristic Algorithms

## 3.1 Building a Solution: Heuristic Frameworks

In this section we recall several heuristics in the literature for related problems. We have adapted them for total weighted tardiness. The first four algorithms ("EST", "HP", "IT' and "GL") are greedy. At each iteration we assume that the machine is available at some time $t$ and we schedule a job in the set $NS$ of unscheduled jobs ($NS$ is initialized with the complete set of jobs). The heuristic stops when all jobs are scheduled, *i.e.*, when set $NS$ is empty (see Algorithm 1). In each algorithm $J_{[i]}$ is the job scheduled in the $i^{th}$ position. The other two algorithms (LA and RBS) can also be seen as "sophisticated" greedy algorithms. Recall that the set of active jobs is dominant for all the studied problems (see

---

**Algorithm 1 Greedy Algorithm**: build a sequence $\sigma = (J_{[1]}, \ldots, J_{[n]})$.

1: $NS \leftarrow \{J_1, \ldots, J_n\}$, $t \leftarrow 0$, $i \leftarrow 0$
2: **while** $NS \neq \emptyset$ **do**
3:    select $J_x \in NS$, $i \leftarrow i + 1$, $J_{[i]} \leftarrow J_x$, $t \leftarrow \max\{t, r_x\} + p_x$, $NS \leftarrow NS - \{J_x\}$
4: **end while**

---

Section 2.1). Let $NS'$ be the set of unscheduled jobs which lead to an active schedule. At

each iteration of the algorithms, $NS'$ is obtained from $NS$ by removing jobs $J_k$ such that $r_k \geq \min_{\{J_i \in NS\}} \{\max\{t, r_i\} + p_i\}$. All the following algorithms use some priority rules which are described at the end of the section.

- Algorithm **EST** (Earliest Start Time) builds non-delay schedules [3],*i.e.*schedules in which the machine can not kept idle if there is a job available for processing: at each iteration the job which can be processed the earliest is chosen from set $NS$. Ties are broken with a priority rule. Note that this algorithm builds active schedules in all cases, since it builds non-delay schedules.

- Algorithm **HP** (Highest Priority) works as follows: At each iteration, a job $J_x$ with maximum priority among those in $NS'$ is chosen (if jobs have the same priority, the one is chosen which can be scheduled the earliest).

- Algorithm **IT** (Insertion Technique) [9]. At each iteration, a job $J_x$ is chosen from $NS$ and scheduled. The job with the highest priority is chosen. We break ties by choosing the job which can be scheduled the earliest. Unlike the other heuristic algorithms, IT can schedule a job which does not lead to an active schedule. We then verify if there are some unscheduled jobs which can be completed before the beginning of $J_x$. If there are such jobs, the one which can be processed the earliest is scheduled, with the priority rule used to break ties. This procedure continues until there is no unscheduled job which can be completed before the beginning of $J_x$. Note that the insertion technique yields an active schedule.

- Algorithm **GL** (Gain - Loss) is a generalization of an algorithm by Chu [8, 10] which was initially described for the total completion time criterion. Two jobs $J_\alpha$ and $J_\beta$ are chosen among unscheduled jobs. Job $J_\alpha$ is chosen from among all unscheduled jobs (set $NS$) using a priority rule. Job $J_\beta$ is chosen using a priority rule from among jobs which are available at time $t$. If we schedule job $J_\beta$ at time $t$ we will eliminate avoidable idle time on the machine. With job $J_\alpha$ scheduled at time $t$ the schedule may be locally optimal if $WT_{\beta\alpha}(t) - WT_{\alpha\beta}(t) \geq 0$. If we schedule job $J_\alpha$ before job $J_\beta$, we may obtain a gain $WT_{\beta\alpha}(t) - WT_{\alpha\beta}(t)$, but we create an idle time of $\max\{t, r_\alpha\} - t$. The gain and the loss entailed by scheduling job $J_\alpha$ is then evaluated. If the gain is at least as large as the loss, we schedule $J_\alpha$; otherwise $J_\beta$ is scheduled.

- Algorithm **LA** (Look-Ahead) is a local search technique by Kanet and Zhou [19]. It defines the alternative courses of action at each decision point by evaluating the consequences of each alternative according to a given criterion and choosing the best alternative. In Algorithm LA, we compute all sequences obtained by putting one of the unscheduled jobs in $NS'$ first and ordering the remaining jobs using a heuristic algorithm (for example EST or HP). After computing the total weighted tardiness of each sequence whith a job scheduled first, we choose the scenario with the minimum total weighted tardiness. Note that this algorithm requires a large amount of CPU time.

6

- Algorithm **RBS** (Recovering Beam Search). It has been described by Della Croce and T'Kindt [14] for the total completion time criterion. Since this method is strongly based on dominance properties valid only for the total completion time criterion, we have not generalized it for the other criteria. In this method, at each iteration, the set $NS$ of unscheduled jobs is filtered using a technique based on dominance properties. Let $NS''$ be this filtered set. For each job $J_i \in NS''$, like in algorithm LA, an evaluation of the cost corresponding to the case, in which $J_i$ is scheduled first after the partial sequence of scheduled jobs, is computed. The difference with respect to LA is that the evaluation is done using a linear combination of lower and upper bounds. The lower bound $LB$ is computed using the $SRPT$ (Shortest Remaining Processing Time) rule on $NS/\{J_i\}$. The upper bound $UB$ is computed using a technique using both greedy algorithms HP and GL (with the priority rule *PRTF*, see Section 3.1) and a filtering method based on the $SRPT$ schedule computed for $LB$. A value $V = (1 - \varphi)LB + \varphi UB$ is computed. At each iteration, the job $J_x$ with the smallest value $V$ is chosen to be scheduled. At the end of each iteration, a recovering step based on an insertion technique is then used to improve the partial schedule. We refer the reader to [14] for more details about this method.

**Priority Rules**

| Name & Ref. | Rule |
|---|---|
| **CPRTWT** New Priority Rule (all criteria) | Combined Priority Rule for Total Weighted Tardiness $\max_{\{J_j \in NS\}}\{\sum_{\{J_l \in NS\}} c_{jl}(t)\}$ with $c_{jl}(t) = 1$ if $WT_{jl}(t) - WT_{lj}(t) \leq 0$, and $c_{jl}(t) = 0$ otherwise. |
| **ATC** [23] $(\sum w_i T_i)$ | Apparent Tardiness Cost $\max_{\{J_j \in NS\}}\{\pi_j = \frac{w_j}{p_j} \exp(\frac{-\max\{0, d_j - t - p_j\}}{2\bar{p}})\}$ $\bar{p} = \sum_{\{J_l \in NS\}} p_l / |NS|$ |
| **X-RM** [20] $(\sum w_i T_i)$ | X-dispatch ATC $\max_{\{J_j \in NS\}}\{\pi_j(1 - \frac{B \max\{0, r_j - t\}}{\tilde{p}})\}$ $B \in \{1.6, 2\}, \tilde{p} \in \{\bar{p}, \min_{\{J_l \in NS\}}\{p_l\}\}$ |
| **COVERT** [23] $(\sum w_i T_i)$ | Weighted Cost Over Time $\max_{\{J_j \in NS\}}\{\frac{w_j}{p_j} \max[0, 1 - \frac{\max\{0, d_j - t - p_j\}}{k p_j}]\}$ |
| **WSPT** [22] $(\sum w_i C_i)$ | Weighted Shortest Processing Time $\min_{J_j \in NS}\{\frac{p_j}{w_j}\}$ |
| **PRTT** [11, 9] $(\sum T_i)$ | Priority Rule for Total Tardiness $\min_{J_j \in NS}\{\max\{t, r_j\} + \max\{d_j, \max\{t, r_j\} + p_j\}\}$ |
| **PRTF** [8, 10] $(\sum C_i)$ | Priority Rule for Total Flow Time $\min_{J_j \in NS}\{2 \max\{t, r_j\} + p_j\}$ |

Table 1: Priority Rules for the total cost problems.

We now present some priority rules which can be used with heuristics (see Table 1). Let $t$ be the time at which the machine is available and let $NS$ be the set of unscheduled jobs at time $t$.

The apparent tardiness cost ATC and the COVERT priority rules [23] are priority rules which combine the Weighted Shortest Processing Time rule and the Minimum Slack rules. They trade off job urgency (slack) against machine utilization. In the X-RM rule, Morton and Ramnath [20] modify the ATC rule to allow heuristic algorithms to insert idle times. A priority correction is made to reduce the priority of late-arriving critical jobs. The PRTT and the PRTF priority rules have been described by Chu [11, 10], for the total tardiness and the total completion time criteria, respectively. These powerful priority rules were derived from local dominance properties. Note that all the previous priority rules are computed in $0(1)$ time for each job and, choosing the best job according to one of these priority rules among $n$ jobs has a cost of $0(n)$ times.

We now define a new priority rule **CPRTWT** (Combined Priority Rule for Total Weighted Tardiness) which uses the local dominance properties. For two jobs $J_j$ and $J_l$, let $c_{jl}(t)$ be a cost which is equal to 1 if $WT_{jl}(t) - WT_{lj}(t) \leq 0$, and to 0 otherwise. We define the priority rule CPRTWT as follows: at time $t$, we choose a job with the maximum value $\sum_{J_l \in NS} c_{jl}(t)$. Note that this priority rule has the great advantage of being valid and applicable to all the studied criteria. Note also that the CPRTWT priority rule is equivalent to the PRTT and the PRTF priority rules when it is used respectively for the total tardiness and the total completion time problem. Indeed, it is easy to prove that in these cases a job chosen with the CPRTWT priority rule is identical to that chosen by the PRTT or the PRTF priority rule. Nevertheless, this new priority rule suffers from the fact that if $i$ jobs have to be compared, then choosing the best job according to the CPRTWT priority rule costs $O(n^2)$. Fortunately, experimental results (see Section 5) show that the CPRTWT priority rule gives very good results compared with the other priority rules (except for PRTT and PRTF, which are equivalent).

## 3.2 Improving a Solution

In this section we describe two procedures which use our dominance rules (see Section 2) and are capable of drastically improving the quality of sequences built using the heuristic algorithms described in the previous section. From now on, let $\sigma = (J_{[1]}, \ldots, J_{[y]})$ be a partial sequence of $y$ scheduled jobs obtained by one of the greedy heuristic algorithms.

---

**Algorithm 2 - MakeLOWSActive**$((J_{[1]}, \ldots, J_{[y]}))$: Make partial sequence $(J_{[y-1]}, J_{[y]})$ *LOWS*-Active.

---
1: **if** $y \geq 3$ **then** $t \leftarrow C_{[y-2]}$ **else** $t \leftarrow 0$ **end if**
2: **if** $y \geq 2$ **and** $\{[WT_{[y]}(t) - WT_{[y-1]}(t) < 0$ **and** $\max\{t, r_{[y]}\} \leq \max\{t, r_{[y-1]}\}]$ **or**
   $[WT_{[y]}(t) - WT_{[y-1]}(t) = 0$ **and** $\max\{t, r_{[y]}\} < \max\{t, r_{[y-1]}\}]\}$ **then**
3:   interchange $J_{[y-1]}$ and $J_{[y]}$
4: **end if**

---

A first improving algorithm allows us to make the partial sequence $\sigma$ *LOWS*-Active (see Section 2.1) on the last two jobs, possibly by interchanging these jobs (see Algorithm 2). In line 2 we check whether the conditions for a *LOWS*-Active schedule are met regarding the last two jobs of the partial sequence. If not, these last two jobs are interchanged (line 3). Note that this algorithm runs in $O(1)$. From now on, we refer to this algorithm as "MakeLOWSActive".

---

**Algorithm 3 - MakeBetter**$((J_{[1]}, \ldots, J_{[y]}))$.
Improve sequence $(J_{[1]}, \ldots, J_{[y]})$ with insertions and interchanges.

---

1:  $x \leftarrow y - 1$
2:  **while** $x \geq 1$ and $r_{[y]} < r_{[x]} + p_{[x]}$ **do**
3:     **for** $i = 1$ to $x - 1$ **do** $J_{(i)} \leftarrow J_{[i]}$, $J_{\{i\}} \leftarrow J_{[i]}$ **end for**
4:     $J_{(x)} \leftarrow J_{[y]}$, MakeLOWSActive$((J_{(1)}, \ldots J_{(x)}))$
5:     $J_{\{x\}} \leftarrow J_{[y]}$, MakeLOWSActive$((J_{\{1\}}, \ldots J_{\{x\}}))$
6:     $J_{\{x+1\}} \leftarrow J_{[x]}$, MakeLOWSActive$((J_{\{1\}}, \ldots J_{\{x+1\}}))$
7:     **for** $i = x + 1$ to $y - 1$ **do**
8:        $J_{(i)} \leftarrow J_{[i]}$, MakeLOWSActive$((J_{(1)}, \ldots J_{(i)}))$
9:        $J_{\{i+1\}} = J_{[i]}$, MakeLOWSActive$((J_{\{1\}}, \ldots J_{\{i\}}))$
10:    **end for**
11:    $J_{(y)} \leftarrow J_{[x]}$, MakeLOWSActive$((J_{(1)}, \ldots J_{(y)}))$
12:    $S \leftarrow (J_{[1]}, \ldots, J_{[y]})$
13:    **if** $(J_{(1)}, \ldots J_{(y)})$ is "better" than $S$ **then** $S \leftarrow (J_{(1)}, \ldots J_{(y)})$ **end if**
14:    **if** $(J_{\{1\}}, \ldots J_{\{y\}})$ is "better" than $S$ **then** $S \leftarrow (J_{\{1\}}, \ldots J_{\{y\}})$ **end if**
15:    replace $(J_{[1]}, \ldots, J_{[y]})$ by $S$, $x \leftarrow x - 1$
16: **end while**

---

A second improving algorithm involves searching a "better" (see Section 2.2) sequence than $\sigma$ (see Algorithm 3). To this end, in lines $3 - 11$, we enumerate the permutations that are obtained from $\sigma$ by inserting the last job $J_{[y]}$ somewhere inside $\sigma$, or by interchanging $J_{[y]}$ with another job in $\sigma$. Let $(J_{\{1\}}, \ldots, J_{\{y\}})$ be the sequence obtained by such an insertion and let $J_{\{i\}}$ be the job in $i$th position in this sequence. In addition, let $(J_{(1)}, \ldots J_{(y)})$ be the sequence obtained by such an interchange and let $J_{(i)}$ be the job in $i$th position in this sequence.

If a "better" sequence is found as a result of the above interchanges or insertions, the current partial sequence $\sigma$ is replaced by this better sequence (see lines $12 - 15$). Since there are $O(|\sigma|)$ such permutations of $\sigma$, and since the comparison of two sequences can be done in a linear time, the algorithm runs in $O(|\sigma|^2)$. To improve the quality of permutations built during the execution of the algorithm, we try to ensure that local optimality holds between each pair of adjacent jobs: during the construction of a sequence, two adjacent jobs can be interchanged to obtain *LOWS*-Active schedules by using the first improving algorithm. Note that this method does not lead necessarily to *LOWS*-Active schedules. From now on, we refer to this algorithm as "MakeBetter".

Either of these two improving algorithms may easily be included in the heuristic al-

gorithms described in Section 3.1. We can use one of these at each iteration by adding it to the end of line $4$ of Algorithm 1. Moreover, in the heuristics LA and RBS, these two improving algorithms can be used to perform the evaluations (Algorithms HP and GL with the PRTF priority rule for the heuristic RBS, and algorithms EST, HP, IT or GL with any priority rules for LA).

Recall that the time complexities of priority rules are not all the same. Computing priority CPRTWT for a job requires $O(|NS|)$ time, whereas priority rules ATC, X-RM and COVERT take $O(1)$ time. Moreover, improving algorithms do not have the same complexity. Hence, the complexities of heuristic algorithms depend on both the improving algorithm and the priority rule used. For instance, Algorithms EST, HP and IT, used with priority rules ATC, X-RM and COVERT, run in $O(n^2)$ time if they are used either with no improving algorithm or with MakeLOWSActive. They run in $O(n^3)$ time if they are used with MakeBetter. Experimental results, which show the effectiveness of these two algorithms in improving the solutions built by the greedy algorithms, are provided in Section 5.

# 4   Tabu Search Method

In this section we describe a Tabu Search method which allows us to build good solutions for total cost problems. The components of our method are described in Sections 4.1, 4.2, 4.3 and 4.4. We describe how these components are used in our method in Section 4.5. Note that this method can be greatly enhanced by techniques from the literature specific to the Tabu Search. Here, we wish simply to show that our techniques can be very useful for these kinds of problems.

Tabu Search [15, 16] is a neighborhood search algorithm which starts with a feasible solution $S$ and tries to improve it iteratively. The improving solution $S'$ is found in some neighborhood of $S$. To be efficient, a neighborhood search algorithm has to avoid cycles. The classical technique used in the Tabu Search method is to keep a list $L$ of the latest solutions (or of the latest moves). If a solution (or a move) is in $L$, it is tabu (*i.e.*, forbidden), and it is not investigated until a given number of iterations has been performed. Below we shall describe a variant of this approach which seems more adequate for the total weighted tardiness problem.

A solution is a sequence of jobs $\sigma$. The schedule $S$ associated with $\sigma$ is obtained by scheduling the jobs as early as possible according to the sequence. From now on, let $\sigma(S)$ be the sequence associated with the schedule $S$. As a cost function we use the total weighted tardiness $WT(S)$ of schedule $S$.

## 4.1   Tabu List

We performed a large number of experiments in order to find a technique which avoids as many cycles as possible. One difficulty, inherent to the total (weighted) tardiness problem, is that a lot of solutions are equivalent. In particular, a typical solution is composed of

partial sequences of on-time jobs which can be exchanged under feasibility constraints to obtain solutions with exactly the same cost. This is also true for sequences of tardy jobs. Consequently, we may need to explore a lot of equivalent solutions. Moreover, it is very easy to cycle even where we forbid the last moves or the last solutions. Our objective is to limit such explorations.

To deal with these issues, we propose the following approach. Instead of storing the last moves or the last solutions, we store the last values of the total cost obtained in the last visited solutions. From now on, a solution is "tabu" if its total cost belongs to the Tabu list. Our experiments show that the list needs to be large in order to be efficient.

## 4.2   Neighborhood

An insertion neighborhood is used. A schedule $S'$ is a neighbor of the current solution $S$ if the sequence $\sigma(S')$ can be obtained from $\sigma(S)$ by a single insertion of a job at some earlier position. To converge quickly to active schedules (which are dominant for this problem), we avoid insertions of a job $J_i$ in a position $j$ if its release date $r_i$ is greater than or equal to the completion time of the job in position $j$ in $\sigma(S)$.

Since the size of the complete neighborhood is $O(n^2)$, and since a sequence is evaluated in linear time, visiting the entire neighborhood requires $0(n^3)$. The best solution $S'$ is chosen from among the non-tabu solutions.

Sequence $S'$ may have a cost greater than that of $S$. If the cost function was previously decreasing, we are in a local optimum (according to the neighborhood used), in which case we shall use the intensification described in the next section. Conversely, if the cost function was previously increasing, the move is accepted.

## 4.3   Intensification

We can view the intensification as a larger neighborhood. In this section we describe an intensification method derived from our local dominance rules. This intensification is based on a kind of dynasearch technique (see [12]). While traditional local search algorithms make a single move at each iteration, dynasearch allows a series of "independent" moves to be performed.

First, we describe in Section 4.3.1 an extended neighborhood relying on the dominance rules described in Section 2.1. In Section 4.3.2 we analyze the consequences of a move on an initial sequence. To make use of all the performed computations, we define in Section 4.3.3 the notion of "compatibility" between several moves, which is an adaptation of the notion of "independence" [12]. In Section 4.3.4 we then propose a Dynasearch technique involving several compatible moves at each iteration, and present a $O(n^3)$ time algorithm which allows us to find the best combination of compatible moves according to the computations which have been made.

### 4.3.1 Elementary Improved Moves Using Local Dominance Rules

In this section we enlarge the neighborhood of Section 4.2 and we use dominance rules described in Section 2.1.

All the sequences obtained by inserting a job in another position, or by the interchanging of two jobs, are now investigated. Recall that the subset of the *LOWS*-Active schedules has been proved to be dominant for the one-machine total (weighted) tardiness problem (Section 2). In order to obtain better sequences, whenever we build a new sequence, we try to obtain *LOWS*-Active schedules. Consider a partial current sequence during the building of a new sequence. Each time we add a job to this partial sequence, we shall possibly interchange the two last jobs to make the partial sequence *LOWS*-Active on the two last jobs, using the MakeLOWSActive algorithm (see Section 3.2). Note that this method does not necessarily yield *LOWS*-Active schedules. The size of this new neighborhood is $O(n^2)$ and can be computed in $O(n^3)$ time, since MakeLOWSActive requires $O(1)$ time.

### 4.3.2 Consequences of Improving Moves

Let us now analyze the consequences of an improving move on a schedule. Let $J_{[i]}$ be the job at the $i^{th}$ position in a schedule $S$ and let $\sigma(S) = (J_{[1]}, \ldots, J_{[n]})$ be the initial sequence corresponding to schedule $S$. Suppose that $i < k$, and let $M_{i,k}$ be one of the following three improving moves: (1) an interchange between the two jobs $J_{[i]}$ and $J_{[k]}$, (2) an insertion of job $J_{[i]}$ at position $k$, or (3) an insertion of job $J_{[k]}$ at position $i$. Let $M_{i,k}(S)$ be the schedule which is obtained after the execution of move $M_{i,k}$ on schedule $S$, and let $\sigma(M_{i,k}(S)) = (J_{\{1\}}, \ldots, J_{\{n\}})$ be the sequence corresponding to the schedule $M_{i,k}(S)$. Additionally, let $G_{i,k}(S)$ be the gain with respect to the total cost of schedule $S$ obtained by move $M_{i,k}$, *i.e.*, $G_{i,k}(S) = WT(S) - WT(M_{i,k}(S))$.

Note that jobs $\{J_{[1]}, \ldots, J_{[i-1]}\}$ do not move, and so we have $J_{\{i\}} = J_{[i]}, \forall i \in \{1, \ldots, i-1\}$. Jobs $\{J_{[i]}, \ldots, J_{[k]}\}$ can move, and their cost may be modified. As a result of the move and the use of the MakeLOWSActive algorithm, the completion time of the job at position $k$ in schedule $S$ may not be the same as the completion time of the job in position $k$ in schedule $M_{i,k}(S)$, *i.e.*, it may happen that $C_{[k]} \neq C_{\{k\}}$. If $C_{[k]} < C_{\{k\}}$, then the jobs $\{J_{[k+1]}, \ldots, J_{[n]}\}$ are scheduled later in $M_{i,k}(S)$, and their cost may increase. If $C_{[k]} > C_{\{k\}}$, then the jobs $\{J_{[k+1]}, \ldots, J_{[n]}\}$ can in certain cases be scheduled earlier in $M_{i,k}(S)$, and their cost may decrease. Note that in these two cases MakeLOWSActive can modify the positions of the jobs to try to make the sequence *LOWS*-Active. Finally, if $C_{[k]} = C_{\{k\}}$, the jobs $\{J_{[k+1]}, \ldots, J_{[n]}\}$ do not move and their cost does not change.

It can be observed that the global gain $G_{i,k}(S)$ is due to the moves of jobs $\{J_{[i]}, \ldots, J_{[k]}\}$, and possibly to the moves of jobs $\{J_{[k+1]}, \ldots, J_{[n]}\}$ if $C_{[k]} \neq C_{\{k\}}$. Thus, let $PG_{[i,k]}(S)$ be the partial gain due to the moves of jobs $\{J_{[i]}, \ldots, J_{[k]}\}$, and let $PG_{[k+1,n]}(S)$ be the partial gain due to the moves of jobs $\{J_{[k+1]}, \ldots, J_{[n]}\}$. Note that values $PG_{[i,k]}(S)$ and $PG_{[k+1,n]}(S)$ can be positive, null or negative. However, $G_{i,k}(S) = PG_{[i,k]}(S) + PG_{[k+1,n]}(S)$ will be positive, since by hypothesis the move is an improving move.

### 4.3.3 Compatibility between two moves

We now define the notion of "compatibility" between two moves, which is an adaptation to our problem of the concept of "independence" described by [12].

Let $M_{i,k}$ et $M_{j,l}$ be two elementary moves which allow us to improve a schedule $S$ (*i.e.*, $G_{i,k}(S) \geq 0$ and $G_{j,l}(S) \geq 0$). Informally, we say that $M_{i,k}$ is compatible with $M_{j,l}$ when we are sure that they can be both performed on the schedule, *i.e.*, the consequences of the first move do not impact on the second.

A first condition for the compatibility of moves $M_{i,k}$ and $M_{j,l}$ is that the two moves are "independent" ([12]), *i.e.*, $k < j$ or $l < i$. If this condition does not hold, the gain from one move can disturb the other. Suppose now that $M_{i,k}$ and $M_{j,l}$ are two independent moves. Without loss of generality, suppose that $k < j$ (*i.e.*, $M_{i,k}$ is further to the left than $M_{j,l}$). Note that if we have $C_{[k]} > C_{\{k\}}$, the jobs on the right of the position $k$ in $S$ are delayed, *i.e.*, the move $M_{i,k}$ disturbs the move $M_{j,l}$. It follows that a second condition for the compatibility of moves $M_{i,k}$ and $M_{j,l}$ is that $C_{[k]} \leq C_{\{k\}}$, and we must avoid any move of jobs on the right of the position $k$ if we perform move $M_{i,k}$ accompanied by move $M_{j,l}$.

In this case the gain to be taken into account is the partial gain $PG_{[i,k]}(S)$ and not the global gain $G_{i,k}(S)$. It is clear that a move $M_{i,k}$ such that $PG_{[i,k]}(S) < 0$ is not considered as an improving move for $S$ if it is accompanied by move $M_{j,l}$. Nevertheless, note that $M_{j,l}$ can be an improving move $S$ if $PG_{[i,k]}(S) < 0$, $PG_{[k+1,n]}(S) > 0$ and $G_{i,k}(S) \geq 0$. We can now define compatibility between two jobs:

**Definition 4.** *Let $S$ be a schedule and let $M_{i,k}$ and $M_{j,l}$ be two improving and independent moves for $S$ such that we have $k < j$ and $PG_{[i,k]}(S) \geq 0$. The moves $M_{i,k}$ and $M_{j,l}$ are compatible if $C_{[k]} \leq C_{\{k\}}$.*

This notion of "compatibility" can be generalized to a combination of more than two independent moves. By extension of the above, for each move $M_{i,k}$ other than the right-most move in the schedule, we cannot delay jobs after index $k$.

**Definition 5.** *Let $E = \{M_{i_1,k_1} \ldots, M_{i_x,k_x}\}$ be a set of $x$ elementary improving and independent moves such that $k_1 < i_2$, $k_2 < i_3$, ..., $k_{x-1} < i_x$. The set $E$ is a combination of compatible moves if for all $j < x$, we have $C_{[k_j]} \leq C_{\{k_j\}}$ and $PG_{[i_j,k_j]}(S) \geq 0$.*

### 4.3.4 Finding the Best Combination of Improving Compatible Moves

The method described in [12] allows the authors to find the "best" combination of "independent" moves. Nevertheless, the notion of "compatibility", which is adapted to the problem with release dates in which one move can disturb a subsequent one (see previous section), does not allow us to find easily the best combination of "compatible" moves. That is why we refer only to a Dynasearch technique [12] which allows us to find a "good" combination of elementary compatible moves.

To this end we build the following valued graph $G$. Two vertices $s$ and $p$ are built. For each possible improving move $M_{i,k}$, we build a vertex $M_{i,k}^1$ if $C_{[k]} \leq C_{\{k\}}$ and $PG_{[i_j,k_j]}(S) \geq 0$, an edge of value $0$ between vertex $s$ and $M_{i,k}^1$, and an edge of value

$PG_{[i_j,k_j]}(S)$ between vertex $M_{i,k}^1$ and vertex $p$. Each of these vertices corresponds to the case in which the move $M_{i,k}$ is made without modification to the positions of jobs after index $k$. We also build a vertex $M_{i,k}^2$, an edge of value $0$ between vertices $s$ and $M_{i,k}^2$, and an edge of value $G_{i,k}(S)$ between vertices $M_{i,k}^2$ and $p$. Each of these vertices corresponds to the case where move $M_{i,k}$ is made and where the algorithm MakeLOWSActive allows us to modify the positions of jobs after the index $k$. Note that this move is permitted only if it is the last move of the combination. All of these edges from $M_{i,k}^2$ are therefore connected to vertex $p$. Finally, for each pair of compatible moves $M_{i,k}$ and $M_{j,l}$ such that $k < j$, we build an edge from vertex $M_{i,k}^1$ to vertex $M_{j,l}^1$, and an edge from vertex $M_{i,k}^1$ to vertex $M_{j,l}^2$, these two vertices being of value $PG_{[i,k]}(S)$.

For any path in $G$ from vertex $s$ to vertex $p$ corresponds a combination of compatible moves belonging to the path. Moreover, the value of a path corresponds to the minimum gain obtained by performing the whole combination of these moves.

We shall obtain the best combination of moves, with respect to this minimum gain, by seeking the longest path between $s$ and $p$ in this valued graph. Let $S$ be a schedule. For each job $J_{[i]}$ in $S$, there are at most $O(n)$ possible interchanges and insertions. There are $O(n^2)$ vertices and $O(n^3)$ edges. The graph is acyclic, and so the longest valued path between $s$ and $p$ can be computed in $O(n^3)$ by dynamic programming.

Note that the described method gives the best combination of compatible moves with respect to a minimum gain (*i.e.*, performing the moves could lead to a better gain than the computed one), unlike [12], whose method yields the best combination of independent moves with a computation of the real gain.

Once this longest path is computed, each move belonging to the path is performed. The process is iterated until no further improvement on the total cost function can be achieved. This intensification may lead to a solution whose cost belongs to the Tabu list. In such a case, we use the diversification procedure described in the next section.

## 4.4 Diversification

To diversify the solution, we define a new cost function $f$ as a linear combination of total weighted tardiness and total earliness costs, *i.e.*, $f(S) = \nu_1 \sum w_i T_i + \nu_2 \sum w_i E_i$, where $E_i = \max\{0, d_i - C_i(S)\}$ is the earliness of job $J_i$ in schedule $S$. Note that other linear combinations such as $f(S) = \nu_1 \sum w_i T_i + \nu_2 T_{max}$, where $T_{max} = \max\{T_i\}$, may be considered. We prefer $\sum w_i E_i$ to $T_{max}$, because many different solutions may all have the same value $T_{max}$.

Parameters $\nu_1$ and $\nu_2$ are randomly generated according to the size of the problem. At the beginning of this procedure, a job $J_i$ is chosen randomly. We consider all the neighbors obtained by interchanging $J_i$ with another job of the current schedule $S$. Among the non-tabu sequences (with respect to the main cost function, *i.e.*, the total weighted tardiness), we choose the interchange with a job $J_k$ which leads to a sequence minimizing the cost function $f$. At the next iteration, the job $J_k$ takes the role of job $J_i$. We iterate a number of times randomly (according to the size of the instance). In practice, this method allows us to explore a new area of the set of feasible solutions, without loosing too much of the

structure of the initial solution.

## 4.5  Global Method

We now show how the above techniques are used in our Tabu search (Algorithm 4). An

---

**Algorithm 4 Tabu Search**

---

 1: Compute an initial solution $S$.
 2: $i \leftarrow 0$
 3: **while** $i \leq$ NbItMax and $time \leq$ timeMax **do**
 4:     Select the best non-tabu neighbor $S'$ of solution $S$.
 5:     **if** $WT(S') > WT(S)$ and we are in a local optimum **then**
 6:         $S' \leftarrow$ solution built by the Intensification.
 7:         **if** $WT(S') \in TabuList$ **then**
 8:             $S' \leftarrow$ solution built by the Diversification.
 9:         **end if**
10:     **end if**
11:     $S \leftarrow S'$.
12:     Add $WT(S)$ in $TabuList$.
13:     **if** the best solution has not been improved since NbItWithoutImp **then**
14:         $S \leftarrow$ solution built randomly.
15:     **end if**
16: **end while**

---

initial solution is built (using, for example, one of the greedy algorithms described in Section 3). We fix a maximum number of iterations nbItMax and a maximum computing time timeMax. The Tabu stops if the total weighted tardiness is equal to $0$ (*i.e.*, the obtained solution is optimal). At each iteration, a solution $S'$ is visited in accordance with the methods described in Sections 4.2,4.3 and 4.4.

In line 4, a sequence $S'$ is first computed with the neighborhood described in Section 4.2. Solution $S'$ may have a cost greater than the cost of $S$. If the cost function was previously decreasing, we are in a local optimum (according to the neighborhood used), and we use the intensification described in the Section 4.3 (see line $6$). Conversely, if the cost function was previously increasing, the move is accepted.

Where intensification leads to a solution whose cost belongs to the Tabu list we use the diversification described in Section 4.4 (line $8$).

During the search, the best obtained solution is stored. Throughout the search, if the best solution has not been improved during a fixed number NbItWithoutImp of iterations, a new solution is randomly built, as in the initialization (line $14$).

# 5  Experimental Results

In this section we provide experimental results demonstrating the effectiveness of all the algorithms described in this paper. All experimental results were obtained using a Pentium IV 2.6 GHz running Windows XP.

In order to be able to compare our results to previous relevant experimental studies, instances were generated in line with the most standard schemes in the literature. For $1|r_i| \sum T_i$ and $1|r_i| \sum w_i T_i$, we use the schemes by Chu [9] and Akturk and Ozdemir [1]. Each instance is generated randomly from uniform distributions of $r_i$, $p_i$ and $d_i$. $p_i$ are uniformly distributed on $[1, 10]$. The $r_i$ and $d_i$ depend on 2 parameters: $\pi_1$ and $\pi_2$. $r_i$ are uniformly distributed on $[0, \pi_1 \sum p_i]$ and $d_i - (r_i + p_i)$ are uniformly distributed on $[0, \pi_2 \sum p_i]$. In the weighted case, $w_i$ are uniformly distributed on $[1, 10]$. Four values for $\pi_1$ and three values for $\pi_2$ are combined to produce 12 instances sets, each containing 20 instances of $n$ jobs, $n \in \{10, \ldots, 200\}$. We then obtain 240 instances for each size $n$.

For $1|r_i| \sum C_i$ and $1|r_i| \sum w_i C_i$, the instances are generated using the same scheme as the test problems of Hariri and Potts [17], and Belouadah, Posner and Potts [5]. For each job, we generate a processing time from the uniform distribution in $[1, 100]$ and a weight from the uniform distribution in $[1, 10]$. For a problem size $n \in \{10, \ldots, 200\}$, an integer release date for each job was generated from the uniform distribution $[0, 50.5nR]$, where $R$ controls the range of the distribution. For each selected value of $n$, 20 problems were generated for each of the $R$ values $0.2, 0.4, 0.6, 0.8, 1.0, 1.25, 1.5, 1.75, 2.0$ and $3.0$ producing 200 problems for each value of $n$.

## 5.1  Results for Greedy Algorithms

We tested the efficiency of the "improving" algorithms (MakeLOWSActive and Make-Better, Section 3.2). Tests were run with greedy algorithms (EST, HP, IT, GL, RBS and LA, Section 3.1), both with and without an improving procedure. It is difficult to analyze the results. Not all heuristics require the same amount of time: the complexity of building a solution depends on the heuristic in question, on the priority rule and on the improving algorithm used in the heuristic. Therefore, a heuristic which gives the best results often requires significantly more computing time (see for instance Table 7).

We first describe the results obtained by the greedy algorithms. In the first table we present the results obtained with algorithms EST, HP, IT and GL. In the second one we present the results obtained by the algorithm LA. A third table is also provided for the total completion time criterion, presenting the results obtained with the RBS algorithm, which is specific to this criterion. In this section we present results only for those instance sizes for which all the optima are known ([18]). Results for larger instance sizes are provided in Section 5.3.

For each heuristic, and for different priority rules according to the studied criterion, we show the average relative gap with respect to the optimum ("gap"), and the relative number of times the optimum is found ("%") over the generated instances for a given instance size $n$. Since the total costs of schedules are often very large for the total weighted

| | | | EST | | | HP | | | IT | | | GL | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $n$ | no | ML | MB | no | ML | MB | no | ML | MB | no | ML | MB |
| $\sum w_i T_i$ | **CPRTWT** | 10 | 0.407 | 0.400 | 0.137 | 0.198 | 0.166 | 0.060 | 0.208 | 0.167 | 0.064 | 0.268 | 0.268 | 0.106 |
| | gap | 20 | 0.497 | 0.477 | 0.240 | 0.481 | 0.334 | 0.108 | 0.441 | 0.305 | 0.106 | 0.363 | 0.349 | 0.204 |
| | | 30 | 2.435 | 2.414 | 1.605 | 1.211 | 1.142 | 0.661 | 0.770 | 0.542 | 0.094 | 2.089 | 2.089 | 1.458 |
| | | 10 | 41 | 44 | 68 | 54 | 59 | 80 | 54 | 59 | 80 | 56 | 56 | 73 |
| | % | 20 | 25 | 27 | 46 | 33 | 38 | 57 | 34 | 39 | 59 | 29 | 29 | 50 |
| | | 30 | 13 | 14 | 28 | 23 | 24 | 41 | 27 | 30 | 47 | 17 | 17 | 33 |
| | **ATC** | 10 | 0.526 | 0.396 | 0.144 | 1.081 | 0.565 | 0.208 | 1.115 | 0.650 | 0.268 | 0.268 | 0.268 | 0.082 |
| | gap | 20 | 0.575 | 0.450 | 0.231 | 1.704 | 0.919 | 0.458 | 1.709 | 0.795 | 0.432 | 0.388 | 0.373 | 0.206 |
| | | 30 | 3.218 | 2.894 | 1.605 | 3.425 | 2.047 | 0.570 | 3.169 | 1.714 | 0.426 | 2.150 | 2.149 | 1.460 |
| | | 10 | 33 | 43 | 69 | 19 | 40 | 66 | 21 | 42 | 68 | 54 | 54 | 72 |
| | % | 20 | 17 | 23 | 42 | 10 | 21 | 39 | 9 | 20 | 38 | 29 | 29 | 49 |
| | | 30 | 10 | 14 | 26 | 10 | 15 | 29 | 10 | 20 | 33 | 17 | 17 | 34 |
| | **XRM** | 10 | 0.526 | 0.396 | 0.144 | 1.200 | 0.610 | 0.110 | 1.589 | 0.880 | 0.298 | 0.268 | 0.268 | 0.082 |
| | gap | 20 | 0.575 | 0.450 | 0.231 | 1.364 | 0.686 | 0.292 | 1.416 | 0.674 | 0.388 | 0.388 | 0.373 | 0.206 |
| | | 30 | 3.218 | 2.894 | 1.605 | 3.927 | 3.011 | 1.595 | 3.989 | 3.134 | 1.652 | 2.151 | 2.151 | 1.461 |
| | | 10 | 33 | 43 | 69 | 25 | 42 | 68 | 20 | 36 | 60 | 54 | 54 | 72 |
| | % | 20 | 17 | 23 | 42 | 12 | 21 | 39 | 11 | 20 | 33 | 29 | 29 | 49 |
| | | 30 | 10 | 14 | 26 | 5 | 13 | 27 | 7 | 12 | 23 | 17 | 17 | 34 |
| | **COVERT** | 10 | 0.634 | 0.448 | 0.169 | 0.838 | 0.307 | 0.122 | 0.874 | 0.313 | 0.130 | 0.266 | 0.266 | 0.082 |
| | gap | 20 | 0.725 | 0.523 | 0.247 | 0.996 | 0.543 | 0.217 | 1.062 | 0.622 | 0.259 | 0.377 | 0.362 | 0.205 |
| | | 30 | 3.515 | 2.391 | 1.666 | 3.154 | 1.925 | 1.372 | 5.700 | 2.337 | 1.360 | 2.113 | 2.112 | 1.462 |
| | | 10 | 26 | 41 | 65 | 23 | 50 | 73 | 23 | 49 | 70 | 54 | 54 | 72 |
| | % | 20 | 10 | 19 | 38 | 10 | 22 | 44 | 9 | 20 | 41 | 29 | 29 | 50 |
| | | 30 | 10 | 15 | 29 | 11 | 19 | 31 | 12 | 20 | 33 | 17 | 17 | 33 |
| $\sum w_i C_i$ | **CPRTWT** | 20 | 13.722 | 13.563 | 5.827 | 5.020 | 5.020 | 3.052 | 5.020 | 5.020 | 3.052 | 10.488 | 10.488 | 5.338 |
| | | 40 | 6.761 | 6.688 | 3.676 | 5.620 | 5.620 | 2.413 | 5.620 | 5.620 | 2.413 | 5.254 | 5.218 | 3.396 |
| | gap | 60 | 4.748 | 4.735 | 2.786 | 6.378 | 6.378 | 2.970 | 6.378 | 6.378 | 2.974 | 4.044 | 4.041 | 2.621 |
| | | 80 | 3.671 | 3.648 | 2.641 | 5.965 | 5.965 | 2.440 | 5.978 | 5.978 | 2.448 | 3.283 | 3.261 | 2.436 |
| | | 100 | 2.873 | 2.860 | 2.155 | 6.324 | 6.323 | 2.690 | 6.347 | 6.346 | 2.668 | 2.618 | 2.606 | 2.054 |
| | | 20 | 16 | 16 | 29 | 43 | 43 | 53 | 43 | 43 | | 31 | 31 | 38 |
| | | 40 | 1 | 1 | 6 | 17 | 17 | 26 | 17 | 17 | 26 | 8 | 8 | 12 |
| | % | 60 | 1 | 1 | 2 | 9 | 9 | 18 | 9 | 9 | 18 | 5 | 5 | 8 |
| | | 80 | 0 | 0 | 0 | 11 | 11 | 15 | 11 | 11 | 15 | 1 | 1 | 1 |
| | | 100 | 0 | 0 | 0 | 5 | 5 | 6 | 5 | 5 | 6 | 1 | 1 | 1 |
| | **SWPT** | 20 | 13.818 | 13.818 | 5.974 | 53.272 | 40.268 | 12.338 | 25.872 | 18.986 | 6.954 | 14.495 | 14.495 | 6.666 |
| | | 40 | 6.752 | 6.752 | 3.674 | 40.145 | 31.666 | 8.775 | 16.839 | 11.973 | 5.391 | 11.326 | 11.304 | 5.459 |
| | gap | 60 | 4.854 | 4.854 | 2.882 | 37.691 | 29.442 | 7.914 | 11.274 | 7.883 | 3.756 | 10.761 | 10.757 | 5.882 |
| | | 80 | 3.724 | 3.724 | 2.651 | 33.964 | 26.967 | 6.938 | 8.249 | 6.781 | 3.223 | 11.879 | 11.874 | 7.280 |
| | | 100 | 2.907 | 2.907 | 2.160 | 31.194 | 24.634 | 6.858 | 7.327 | 5.604 | 2.835 | 10.792 | 10.787 | 6.838 |
| | | 20 | 16 | 16 | 29 | 1 | 8 | 29 | 6 | 12 | 24 | 30 | 30 | 36 |
| | | 40 | 1 | 1 | 6 | 0 | 4 | 9 | 1 | 1 | 5 | 5 | 5 | 11 |
| | % | 60 | 1 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 1 | 4 | 4 | 6 |
| | | 80 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 1 | 1 | 1 |
| | | 100 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| $\sum T_i$ | **PRTT** | 10 | 0.163 | 0.163 | 0.024 | 0.087 | 0.086 | 0.051 | 0.095 | 0.094 | 0.052 | 0.028 | 0.028 | 0.016 |
| | | 20 | 0.174 | 0.174 | 0.111 | 0.197 | 0.158 | 0.078 | 0.182 | 0.142 | 0.076 | 0.187 | 0.187 | 0.119 |
| | gap | 30 | 0.138 | 0.138 | 0.079 | 0.244 | 0.208 | 0.058 | 0.339 | 0.177 | 0.045 | 0.174 | 0.174 | 0.114 |
| | | 40 | 0.181 | 0.181 | 0.145 | 0.416 | 0.273 | 0.050 | 0.411 | 0.300 | 0.075 | 0.236 | 0.236 | 0.182 |
| | | 50 | 0.297 | 0.297 | 0.132 | 0.204 | 0.190 | 0.037 | 0.180 | 0.149 | 0.027 | 0.339 | 0.339 | 0.185 |
| | | 10 | 75 | 75 | 85 | 78 | 78 | 86 | 78 | 78 | 86 | 82 | 82 | 89 |
| | | 20 | 57 | 57 | 66 | 51 | 53 | 67 | 53 | 55 | 67 | 53 | 53 | 63 |
| | % | 30 | 41 | 41 | 51 | 42 | 45 | 61 | 48 | 50 | 63 | 40 | 40 | 51 |
| | | 40 | 39 | 39 | 51 | 39 | 44 | 59 | 44 | 48 | 58 | 33 | 33 | 46 |
| | | 50 | 30 | 30 | 43 | 36 | 38 | 55 | 43 | 45 | 57 | 28 | 28 | 38 |
| $\sum C_i$ | **PRTF** | 20 | 4.076 | 4.076 | 2.086 | 6.404 | 6.404 | 3.179 | 6.404 | 6.404 | 3.179 | 2.380 | 2.380 | 1.857 |
| | | 40 | 1.992 | 1.992 | 1.465 | 9.037 | 9.037 | 4.187 | 9.037 | 9.037 | 4.187 | 1.661 | 1.661 | 1.354 |
| | gap | 60 | 1.351 | 1.351 | 1.133 | 9.743 | 9.743 | 3.862 | 9.743 | 9.743 | 3.862 | 1.087 | 1.087 | 0.992 |
| | | 80 | 0.893 | 0.893 | 0.754 | 9.014 | 9.014 | 3.235 | 9.014 | 9.014 | 3.235 | 0.733 | 0.733 | 0.661 |
| | | 100 | 0.726 | 0.726 | 0.628 | 9.493 | 9.493 | 3.487 | 9.493 | 9.493 | 3.487 | 0.621 | 0.621 | 0.564 |
| | | 20 | 28 | 28 | 48 | 46 | 46 | 58 | 46 | 46 | 58 | 55 | 55 | 64 |
| | | 40 | 12 | 12 | 21 | 24 | 24 | 27 | 24 | 24 | 27 | 29 | 29 | 33 |
| | % | 60 | 4 | 4 | 6 | 12 | 12 | 18 | 12 | 12 | 18 | 12 | 12 | 14 |
| | | 80 | 1 | 1 | 2 | 6 | 6 | 8 | 6 | 6 | 8 | 2 | 2 | 3 |
| | | 100 | 1 | 1 | 3 | 5 | 5 | 7 | 5 | 5 | 7 | 4 | 4 | 6 |

Table 2: Results of algorithms EST, HP, IT and GL.

completion time and the total completion time criteria, the relative gaps have been multiplied by 1000 for these two criteria. In each case the column "no" gives the results obtained with the heuristics alone. "ML" gives the results obtained with heuristics used together with the MakeLOWSActive algorithm. Finally, "MB" gives the results obtained with heuristics used together with the MakeBetter algorithm.

In Table 2 we compare the heuristics EST, HP, IT and GL for each criterion with different priority rules: CPRTWT, ATC, X-RM and COVERT for the total weighted tardiness, CPRTWT and SWPT for the total weighted completion time, PRTT for the total tardiness and PRTF for the total completion time. We can see the effectiveness of the MakeLOWS-Active and MakeBetter algorithms in improving the quality of solutions. We can also see that the new priority rule CPRTWT is very efficient compared to the other priority rules for the $1|r_i|\sum w_i T_i$ and $1|r_i|\sum w_i C_i$ problems. Indeed, most of the time it gives the best results among heuristics algorithms used with or without improving algorithms.

For $1|r_i|\sum w_i T_i$, the best results are obtained with heuristic IT using algorithm Make-Better and the priority rule CPRTWT. For $1|r_i|\sum w_i C_i$, the best results are obtained with heuristic HP using algorithm MakeBetter and the priority rule CPRTWT. For $1|r_i|\sum T_i$, note that algorithms EST and GL do not need algorithm MakeLOWSActive, since these heuristics, used with priority rule PRTT, build schedules which are already *LOWS*-Active. For $1|r_i|\sum C_i$, note that heuristic IT gives exactly the same results as heuristic HP. This is because the priority rule PRTF strongly depends on the release dates of the jobs. Consequently it is never possible to insert a job before a job which is chosen according to the priority rule PRTF (see heuristic IT in Section 3.1). Note that heuristics EST, HP and GL build schedules which are always *LOWS*-Active. Therefore it is not useful to use algorithm MakeLOWSActive to improve the solutions for this criterion. On the other hand, the algorithm MakeBetter is able to improve the solutions for all algorithms. The best results are obtained with heuristic GL using algorithm MakeBetter. Note that heuristic HP (or IT) frequently yields the optimum. It would, however, appear that these heuristics generally give solutions of less good quality.

Recall that for algorithms LA and RBS the improving algorithms can be used in two ways. They can be used at each iteration when scheduling a job. Alternatively, they can be used inside the heuristic algorithm(s), which allows evaluations to be computed for each possible job which can be scheduled at each iteration of the algorithm (EST, HP, IT or GL for algorithm LA, and only HP and GL for algorithm RBS). As for the other heuristics, we report results when LA and RBS are used with no improving algorithm ("no"), with MakeLOWSActive ("ML"), and with MakeBetter ("MB"), at the end of each iteration of LA or RBS. Moreover, in all cases, we report results when an evaluation heuristic H is used with no improving algorithm ("no (H)"), with algorithm MakeLOWSActive ("ML (H)") and with algorithm MakeBetter ("MB (H)").

In Table 3 we provide results obtained with algorithm LA. We only report results for the combination which gives the best results for each of the criteria. For $1|r_i|\sum w_i T_i$ and $1|r_i|\sum w_i C_i$, we use the heuristic HP and the priority rule CPRTWT. For $1|r_i|\sum T_i$, we use the heuristic IT and the priority rule PRTT. Finally, for $1|r_i|\sum C_i$, we use the heuristic GL and the priority rule PRTF. We can see that algorithm LA gives very good

18

| | | no | | | | | | | | ML | | | | | | | | MB | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | no(H) | | ML(H) | | MB(H) | | | no(H) | | ML(H) | | MB(H) | | | no(H) | | ML(H) | | MB(H) | |
| | $n$ | gap | % | gap | % | gap | % | | gap | % | gap | % | gap | % | | gap | % | gap | % | gap | % |
| | 10 | 0.025 | 90 | 0.016 | 92 | 0.003 | 97 | | 0.010 | 94 | 0.009 | 95 | 0.001 | 98 | | 0.004 | 97 | 0.004 | 98 | 0.001 | 99 |
| $\sum w_i T_i$ | 20 | 0.095 | 58 | 0.099 | 63 | 0.025 | 79 | | 0.067 | 63 | 0.074 | 68 | 0.022 | 85 | | 0.039 | 77 | 0.051 | 80 | 0.013 | 92 |
| HP, CPRTWT | 30 | 0.182 | 45 | 0.149 | 48 | 0.027 | 70 | | 0.120 | 53 | 0.102 | 56 | 0.021 | 77 | | 0.054 | 63 | 0.048 | 68 | 0.015 | 83 |
| | 20 | 1.462 | 75 | 1.462 | 75 | 0.868 | 82 | | 1.153 | 80 | 1.153 | 80 | 0.649 | 86 | | 0.465 | 88 | 0.465 | 88 | 0.285 | 92 |
| $\sum w_i C_i$ | 40 | 1.932 | 46 | 1.932 | 46 | 0.856 | 58 | | 1.524 | 49 | 1.524 | 49 | 0.691 | 63 | | 0.779 | 58 | 0.779 | 58 | 0.444 | 72 |
| | 60 | 2.688 | 31 | 2.688 | 31 | 1.111 | 40 | | 2.306 | 35 | 2.306 | 35 | 0.871 | 43 | | 1.379 | 40 | 1.379 | 40 | 0.636 | 46 |
| HP, CPRTWT | 80 | 2.848 | 26 | 2.848 | 26 | 1.250 | 32 | | 2.480 | 29 | 2.480 | 29 | 1.054 | 37 | | 1.576 | 32 | 1.576 | 32 | 0.671 | 44 |
| | 100 | 2.928 | 24 | 2.928 | 24 | 1.282 | 29 | | 2.555 | 28 | 2.554 | 28 | 1.094 | 33 | | 1.631 | 32 | 1.630 | 32 | 0.684 | 36 |
| | 10 | 0.001 | 99 | 0.001 | 99 | 0.001 | 100 | | 0.000 | 99 | 0.000 | 99 | 0.000 | 100 | | 0.000 | 100 | 0.000 | 100 | 0.000 | 100 |
| $\sum T_i$ | 20 | 0.033 | 83 | 0.026 | 84 | 0.013 | 90 | | 0.021 | 87 | 0.022 | 87 | 0.010 | 94 | | 0.008 | 91 | 0.008 | 92 | 0.004 | 96 |
| | 30 | 0.072 | 68 | 0.043 | 69 | 0.016 | 82 | | 0.030 | 74 | 0.034 | 75 | 0.010 | 84 | | 0.017 | 85 | 0.013 | 85 | 0.008 | 90 |
| IT, PRTT | 40 | 0.102 | 64 | 0.085 | 65 | 0.037 | 80 | | 0.065 | 70 | 0.065 | 71 | 0.031 | 85 | | 0.010 | 80 | 0.010 | 81 | 0.002 | 90 |
| | 50 | 0.036 | 63 | 0.025 | 64 | 0.008 | 76 | | 0.023 | 72 | 0.020 | 73 | 0.006 | 81 | | 0.015 | 81 | 0.014 | 82 | 0.003 | 87 |
| | 20 | 0.535 | 85 | 0.535 | 85 | 0.233 | 90 | | 0.482 | 86 | 0.482 | 86 | 0.204 | 91 | | 0.400 | 88 | 0.400 | 88 | 0.172 | 91 |
| $\sum C_i$ | 40 | 0.327 | 68 | 0.327 | 68 | 0.280 | 70 | | 0.253 | 71 | 0.253 | 71 | 0.206 | 74 | | 0.224 | 75 | 0.224 | 75 | 0.168 | 78 |
| | 60 | 0.261 | 51 | 0.261 | 51 | 0.228 | 53 | | 0.214 | 57 | 0.214 | 57 | 0.193 | 58 | | 0.149 | 61 | 0.149 | 61 | 0.134 | 63 |
| GL, PRTF | 80 | 0.221 | 35 | 0.221 | 35 | 0.198 | 39 | | 0.180 | 39 | 0.180 | 39 | 0.160 | 43 | | 0.139 | 42 | 0.139 | 42 | 0.128 | 46 |
| | 100 | 0.204 | 29 | 0.204 | 29 | 0.176 | 34 | | 0.166 | 33 | 0.166 | 33 | 0.147 | 39 | | 0.128 | 40 | 0.128 | 40 | 0.121 | 43 |

Table 3: Results of algorithm LA.

results. With $n = 20$, it yields the optimum $9$ times out of $10$ on average when used with the improving algorithm MakeBetter.

| | no | | | | | | ML | | | | | | MB | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | no (H) | | ML (H) | | MB (H) | | no (H) | | ML (H) | | MB (H) | | no (H) | | ML (H) | | MB (H) | |
| $n$ | gap | % | gap | % | gap | % | gap | % | gap | % | gap | % | gap | % | gap | % | gap | % |
| 20 | 0.541 | 79 | 0.541 | 79 | 0.406 | 82 | 0.541 | 79 | 0.541 | 79 | 0.406 | 82 | 0.519 | 79 | 0.519 | 79 | 0.385 | 82 |
| 40 | 0.372 | 53 | 0.371 | 53 | 0.280 | 60 | 0.372 | 54 | 0.371 | 53 | 0.280 | 60 | 0.350 | 54 | 0.350 | 53 | 0.268 | 61 |
| 60 | 0.307 | 39 | 0.307 | 38 | 0.242 | 43 | 0.307 | 39 | 0.307 | 38 | 0.242 | 43 | 0.296 | 39 | 0.295 | 39 | 0.234 | 44 |
| 80 | 0.322 | 25 | 0.321 | 25 | 0.267 | 28 | 0.322 | 25 | 0.321 | 25 | 0.267 | 28 | 0.324 | 25 | 0.323 | 25 | 0.264 | 28 |
| 100 | 0.263 | 22 | 0.261 | 22 | 0.225 | 26 | 0.263 | 22 | 0.261 | 22 | 0.225 | 26 | 0.270 | 22 | 0.270 | 22 | 0.230 | 26 |

Table 4: $\sum C_i$: results of algorithm RBS.

In Table 4, we provide results obtained with algorithm RBS. Heuristics HP and GL are used with PRTF inside RBS to compute evaluations for each possible job which can be scheduled at each iteration of the algorithm. We can see that algorithm RBS gives very good results particularly when used with the improving algorithm MakeBetter. Nevertheless, the obtained results are slightly less good than those obtained with LA. It should be remembered that this heuristic take less time to execute (see Table 7).

In [14], only one preferred node is retained at each iteration of the RBS procedure in order to minimize the CPU time required by the procedure. Nevertheless, it is interesting to test if a larger beam size can lead to better solutions. Thus, additional tests for different size of beam have been done. This leads to the following remarks. Even with an infinite beam size, the procedures MakeLOWSActive and MakeBetter improve the results obtained by the RBS approach. Even with an infinite beam size, the RBS approach is dominated by the LA approach. Indeed, the crude filter procedure used in the RBS is fast but may result in discarding good solutions. As noticed by [14] branches leading to optimal solutions in the search tree could be pruned in the nodes evaluation process and the recovering step could not be able to repair these situations. Then, even for small size of instances (such as 10), all the optima are not found. Obviously, for large beam sizes, the algorithm is rather slow. In the original RBS approach, the node are not pruned if the

evaluation of the node is greater than the current best found solution. Indeed, the recovering step on this node or on its derived nodes can lead to better solutions. Nevertheless, from size 60 on, the number of nodes to evaluate is too high and the procedure is much too slow. That is why, it is better in this situation to use the evaluations to prune the nodes. Consequently, the results become slightly less good.

## 5.2  Results Obtained by the Tabu Search

| | $n$ | nb. iter. | | nb. int. | | nb. div. | | nb. rest. | | time (ms) | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | mean | max | mean | max | mean | max | mean | max | mean | max |
| CM | 10 | 4 | 13 | 0 | 5 | 0 | 0 | 1 | 1 | 0 | 16 |
| | 20 | 17 | 442 | 1 | 127 | 1 | 124 | 1 | 1 | 1 | 110 |
| | 30 | 35 | 480 | 2 | 69 | 1 | 66 | 1 | 1 | 5 | 219 |
| | 40 | 68 | 2076 | 3 | 146 | 2 | 137 | 1 | 3 | 31 | 1328 |
| | 50 | 99 | 1974 | 5 | 254 | 4 | 247 | 1 | 3 | 88 | 2141 |
| | mean | 45 | 997 | 2 | 120 | 2 | 115 | 1 | 2 | 25 | 763 |
| CM-DR | 10 | 6 | 35 | 0 | 9 | 0 | 0 | 1 | 1 | 0 | 16 |
| | 20 | 24 | 641 | 3 | 152 | 2 | 142 | 1 | 1 | 2 | 125 |
| | 30 | 68 | 2931 | 7 | 510 | 6 | 489 | 1 | 5 | 15 | 1016 |
| | 40 | 153 | 6035 | 12 | 713 | 10 | 653 | 1 | 11 | 84 | 4703 |
| | 50 | 259 | 16137 | 16 | 837 | 14 | 750 | 1 | 31 | 259 | 16063 |
| | mean | 102 | 5156 | 8 | 444 | 6 | 407 | 1 | 10 | 72 | 4385 |
| CM-TLIST | 10 | 4 | 47 | 0 | 18 | 0 | 0 | 1 | 1 | 0 | 16 |
| | 20 | 22 | 2040 | 4 | 807 | 3 | 749 | 1 | 5 | 2 | 250 |
| | 30 | 106 | 9090 | 47 | 8726 | 39 | 7990 | 1 | 39 | 49 | 8125 |
| | 40 | 169 | 8041 | 74 | 6055 | 58 | 4837 | 1 | 15 | 186 | 9922 |
| | 50 | 167 | 7901 | 58 | 4218 | 49 | 3861 | 1 | 11 | 251 | 11500 |
| | mean | 94 | 5424 | 37 | 3965 | 30 | 3487 | 1 | 14 | 98 | 5963 |
| CM-DR-TLIST | 10 | 35 | 9022 | 14 | 4034 | 14 | 1 | 1 | 19 | 1 | 125 |
| | 20 | 261 | 98035 | 108 | 44222 | 98 | 40352 | 1 | 197 | 27 | 10985 |
| | 30 | 573 | 91993 | 241 | 38204 | 200 | 28276 | 2 | 183 | 178 | 29625 |
| | 40 | 491 | 37076 | 198 | 16598 | 165 | 14910 | 2 | 75 | 375 | 26000 |
| | 50 | 424 | 20671 | 160 | 9043 | 137 | 8301 | 2 | 41 | 661 | 29937 |
| | mean | 357 | 51359 | 144 | 22420 | 123 | 18368 | 2 | 103 | 248 | 19334 |

Table 5: $\sum T_i$, Comparing the Efficiency of our Techniques in the Tabu Search.

In this section we analyze the results obtained by our Tabu Search method. We show that techniques described in Section 4 allow us to obtain good results for all criteria. Recall that this method can be greatly enhanced by techniques from the literature specific to the Tabu Search. Here we wish to show that our techniques can be very useful for these kinds of problems. As in the previous section, we provide results only for instance sizes where all the optima are known ([18]). Results for larger instance sizes are given in Section 5.3.

In Table 5 we show the effectiveness of our Tabu Search method and of all our techniques in improving the behavior of the method. This table shows experimental results for the method in relation to the total tardiness criterion. For each $n = \{10, 20, \ldots, 50\}$, the method was executed $5$ times for all the $240$ instances. We arbitrarily set the maximum computing time to $30$ seconds and the Tabu list size to $200$. Moreover, the initial solution is computed randomly. We provide the results when the complete method is run ("CM"), when the dominance rules are not used inside the intensification ("CM-DR"), when the Tabu list is not used ("CM-TLIST"), and when the method is run without dominance rules and without Tabu list ("CM-DR-TLIST"). We provide statistics on the number of iterations ("nb. iter."), the number of intensifications ("nb. int."), the number of diversifications ("nb. div."), the number of restarts ("nb. rest.") and the time (in milliseconds)

| | $n$ | nb. iter. | | nb. int. | | nb. div. | | nb. rest. | | time (ms) | | % | | | gap | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | mean | max | mean | max | mean | max | mean | max | mean | max | min | mean | max | min | mean | max |
| $\sum w_i T_i$ | 10 | 5 | 26 | 0 | 8 | 0 | 6 | 1 | 1 | 0 | 16 | 100 | 100 | 100 | 0.000 | 0.000 | 0.000 |
| | 15 | 16 | 1394 | 3 | 437 | 2 | 423 | 1 | 3 | 1 | 156 | 100 | 100 | 100 | 0.000 | 0.000 | 0.000 |
| | 20 | 18 | 249 | 2 | 75 | 1 | 71 | 1 | 1 | 1 | 94 | 100 | 100 | 100 | 0.000 | 0.000 | 0.000 |
| | 25 | 31 | 563 | 4 | 180 | 3 | 176 | 1 | 1 | 5 | 219 | 100 | 100 | 100 | 0.000 | 0.000 | 0.000 |
| | 30 | 54 | 1812 | 10 | 529 | 9 | 522 | 1 | 3 | 28 | 1234 | 100 | 100 | 100 | 0.000 | 0.000 | 0.000 |
| | mean | 25 | 809 | 4 | 246 | 3 | 240 | 1 | 2 | 7 | 344 | 100 | 100 | 100 | 0.000 | 0.000 | 0.000 |
| $\sum w_i C_i$ | 10 | 6 | 72 | 1 | 23 | 0 | 19 | 1 | 1 | 0 | 16 | 100 | 100 | 100 | 0.000 | 0.000 | 0.000 |
| | 20 | 26 | 1031 | 6 | 383 | 4 | 349 | 1 | 3 | 3 | 125 | 100 | 100 | 100 | 0.000 | 0.000 | 0.000 |
| | 30 | 80 | 2551 | 22 | 1033 | 20 | 998 | 1 | 5 | 55 | 3438 | 100 | 100 | 100 | 0.000 | 0.000 | 0.000 |
| | 40 | 263 | 12962 | 85 | 4505 | 79 | 4261 | 1 | 25 | 530 | 28297 | 98 | 99 | 99 | 0.003 | 0.006 | 0.015 |
| | 50 | 407 | 6572 | 131 | 2405 | 124 | 2259 | 1 | 13 | 1789 | 29688 | 96 | 97 | 98 | 0.050 | 0.063 | 0.094 |
| | 60 | 758 | 9829 | 248 | 3603 | 235 | 3413 | 2 | 19 | 6020 | 59953 | 90 | 91 | 92 | 0.108 | 0.123 | 0.142 |
| | 70 | 603 | 5825 | 183 | 2240 | 173 | 2118 | 1 | 11 | 7848 | 59984 | 81 | 83 | 85 | 0.125 | 0.180 | 0.232 |
| | 80 | 753 | 4962 | 236 | 1847 | 221 | 1765 | 2 | 9 | 13082 | 59953 | 68 | 72 | 74 | 0.232 | 0.341 | 0.461 |
| | 90 | 584 | 3340 | 162 | 1158 | 151 | 1081 | 1 | 5 | 14869 | 59953 | 59 | 63 | 65 | 0.475 | 0.636 | 0.713 |
| | 100 | 588 | 2374 | 161 | 887 | 150 | 817 | 1 | 5 | 19128 | 60125 | 55 | 56 | 59 | 0.835 | 0.872 | 0.924 |
| | mean | 407 | 4952 | 123 | 1808 | 116 | 1708 | 1 | 10 | 6332 | 36153 | 85 | 86 | 87 | 0.183 | 0.222 | 0.258 |
| $\sum C_i$ | 10 | 5 | 52 | 1 | 22 | 0 | 17 | 1 | 1 | 0 | 16 | 100 | 100 | 100 | 0.000 | 0.000 | 0.000 |
| | 20 | 23 | 1364 | 4 | 456 | 3 | 448 | 1 | 3 | 2 | 187 | 100 | 100 | 100 | 0.000 | 0.000 | 0.000 |
| | 30 | 90 | 6509 | 21 | 2106 | 19 | 2006 | 1 | 11 | 40 | 3703 | 100 | 100 | 100 | 0.000 | 0.000 | 0.000 |
| | 40 | 222 | 19654 | 54 | 5779 | 50 | 5451 | 1 | 39 | 266 | 28265 | 99.5 | 99.8 | 100 | 0.000 | 0.002 | 0.004 |
| | 50 | 483 | 14916 | 115 | 5023 | 106 | 4725 | 1 | 29 | 1072 | 29719 | 97 | 98 | 99 | 0.002 | 0.004 | 0.006 |
| | 60 | 802 | 16485 | 155 | 3288 | 143 | 3147 | 2 | 33 | 2982 | 59859 | 98 | 98 | 99 | 0.002 | 0.012 | 0.029 |
| | 70 | 1098 | 19324 | 182 | 3178 | 166 | 2816 | 2 | 33 | 5798 | 59687 | 92 | 94 | 96 | 0.018 | 0.022 | 0.024 |
| | 80 | 1298 | 13820 | 184 | 2085 | 168 | 1964 | 2 | 21 | 9185 | 59875 | 78 | 80 | 81 | 0.068 | 0.134 | 0.206 |
| | 90 | 1363 | 11451 | 166 | 1334 | 150 | 1254 | 2 | 21 | 12417 | 60000 | 69 | 70 | 70 | 0.212 | 0.328 | 0.409 |
| | 100 | 1395 | 9261 | 143 | 1045 | 128 | 973 | 2 | 15 | 16195 | 60219 | 60 | 60 | 61 | 0.860 | 0.991 | 1.100 |
| | mean | 678 | 11284 | 102 | 2432 | 93 | 2280 | 1 | 21 | 4796 | 36153 | 89 | 90 | 90 | 0.116 | 0.149 | 0.178 |

Table 6: $\sum w_i T_i$, $\sum w_i C_i$, $\sum C_i$, Results obtained by the Tabu Search.

needed to converge to the best found solution. For all these statistics, the column "mean" is the average (over the averages for the 5 executions for each instance) over the 240 instances. Moreover, the column "max" is the maximum (over the maxima for the 5 executions for each instance) over the 240 instances. All optima are reached 5 times for each of these instances, in a very short time when the method is used completely ("CM"). Our method reaches the optimum for 100% of the instances for instance sizes lower than or equal to 50. That is why we have not reported the relative number of times the optimum is found, or the average relative gap in relation to the optimum. For $n = 50$ jobs, the average computing time needed and the average number of iterations to find the optimum are respectively 0.09 seconds and 99 iterations. We remark that if the method is used without dominance rules inside the intensification or without the Tabu list, it takes much more time to converge to the optimum. Moreover, in a certain number of cases, the optimum is not found (which has not been reported in the table).

In Table 6 we provide the same statistics for the other criteria (the total weighted tardiness, the total weighted completion time and the total completion time criteria) when the complete method is used. We arbitrarily set the maximum computing time to 30 seconds if $n \leq 50$ and to 60 seconds if $n \geq 60$. Moreover the Tabu list size was set to 200 in all cases. The initial solution is computed randomly. Since not all the optima are found, we also provide the average relative gap ("gap") and the relative number of times that the optimum is found ("%").

For the total weighted tardiness problem, all optima are again reached 5 times for each of these instances, in a very short time. For $n = 35$ jobs, the average computing time needed and the average number of iterations to find the optimum are respectively 0.03

seconds and 54 iterations. For the total weighted completion time and the total completion time criteria, note that all optima are found. The results may easily be compared with the other best methods (*i.e.*, algorithm LA for all criteria, or algorithm RBS for the total completion time criteria). The Tabu Search allows us to find better results in small amount of time. For example, in the case of the total completion time criterion, the Tabu Search method reaches an optimum in 90% of the instances, in an average time of 5 seconds, while the algorithm LA reaches an optimum in 68% of the instances, in 2 seconds in average. Nevertheless, note that the relative gap is larger with the Tabu Search method for $n \geq 90$. The Tabu Search gives the optimum more often, but when the optimum is not reached, the solution is very far from the optimum. This is because the initial solution is computed randomly for the Tabu Search method, and the method does not have sufficient time to find the area where the best solution is located. We can easily obtain better results, either by allocating more than 60 seconds to the Tabu Search method, or by computing the initial solution with a greedy algorithm described in Section 3.1 (see next section).

## 5.3 Comparing Methods for Large Instance Sizes

| | | $n$ | 100 | | 150 | | 200 | | mean | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | cpu (ms) | av. $\bar{F}$ | cpu (ms) | av. $\bar{F}$ | cpu (ms) | av. $\bar{F}$ | cpu (ms) | av. $\bar{F}$ |
| $\sum w_i T_i$ | IT | | 48 | 88627 | 151 | 354675 | 340 | 619552 | 179 | 354285 |
| | IT (MB) | | 71 | 85393 | 232 | 343514 | 544 | 603523 | 282 | 344143 |
| | LA | | 24928 | 85559 | 185392 | 344521 | 746811 | 604136 | 319044 | 344739 |
| | LA (MB) | | 46027 | 83319 | 345019 | 337126 | 1428632 | 591644 | 606559 | 337363 |
| | Tabu | | 38153 | 82904 | 158303 | 334749 | 270292 | 588124 | 155583 | 335259 |
| $\sum w_i C_i$ | HP | | 9 | 1923118 | 26 | 4254697 | 59 | 7477394 | 31 | 4551736 |
| | HP (MB) | | 20 | 1919045 | 68 | 4246210 | 155 | 7463042 | 81 | 4542766 |
| | LA | | 5073 | 1919242 | 39454 | 4247184 | 153181 | 7465708 | 65902 | 4544045 |
| | LA (MB) | | 12105 | 1916779 | 95821 | 4242106 | 375380 | 7458060 | 161102 | 4538982 |
| | Tabu | | 81002 | 1916102 | 185152 | 4241208 | 368397 | 7457137 | 211517 | 4538149 |
| $\sum T_i$ | IT | | 1 | 4634 | 1 | 10305 | 1 | 18188 | 1 | 11042 |
| | IT (MB) | | 6 | 4588 | 16 | 10197 | 36 | 18072 | 20 | 10952 |
| | LA | | 270 | 4584 | 1262 | 10199 | 3824 | 18068 | 1785 | 10950 |
| | LA (MB) | | 4198 | 4561 | 25915 | 10154 | 94400 | 18005 | 41504 | 10907 |
| | Tabu | | 10159 | 4558 | 54606 | 10146 | 150701 | 17996 | 71822 | 10900 |
| $\sum C_i$ | GL | | 23 | 356295 | 77 | 789024 | 182 | 1407782 | 94 | 851034 |
| | GL (MB) | | 28 | 356280 | 91 | 789002 | 217 | 1407743 | 112 | 851008 |
| | BS | | 829 | 356200 | 4453 | 788934 | 13491 | 1407576 | 6258 | 850903 |
| | BS (MB) | | 1345 | 356192 | 7194 | 788908 | 21640 | 1407542 | 10059 | 850880 |
| | LA | | 3382 | 356187 | 22145 | 788909 | 83427 | 1407555 | 36318 | 850884 |
| | LA (MB) | | 5390 | 356166 | 35969 | 788872 | 137124 | 1407495 | 59495 | 850844 |
| | Tabu | | 32907 | 356153 | 33153 | 788888 | 108360 | 1407564 | 58140 | 850868 |

Table 7: Comparing methods for large sizes of instance.

In Table 7 we provide results for large instance sizes. For $n = \{100, 150, 250\}$ and for each criterion, we provide the results obtained by the best algorithm from among EST, HP, IT and GL, the results obtained by LA, and the results obtained by the Tabu Search method. Moreover, for the total completion time criterion, we provide the results obtained by algorithm RBS. For each greedy algorithm, we provide the results obtained by the algorithm used with or without the improving algorithm MakeBetter ("MB"). For each method and for each instance size we give the computing times in milliseconds and the average total cost ("$\bar{F}$") over the generated instances. For the Tabu Search method, we arbitrarily set the maximum computing time to $n/10$ minutes. Moreover the Tabu list

22

size was set to 200 in all cases. The computing time shown for the Tabu Search is the time required to converge to the best found solution.

The best algorithm to use is highly dependent on the time available for seeking a solution. The results show that the use of heuristics such as HP(MB), IT(MB) or GL(MB) is satisfactory if a good solution needs to be obtained quickly. Moreover, the algorithm MakeBetter can drastically improve solutions at an insignificant cost. Indeed, even for $n = 200$ jobs, the computing time needed to obtain a solution is lower than $0.5$ seconds. The algorithm LA used with no improving algorithm would appear to be less interesting, since it gives equivalent or less good solutions in comparison to the other greedy heuristics (HP(MB), IT(MB) or GL(MB)), and with significantly more computing effort. If more computing time is available, the Tabu Search method appears to be the most interesting algorithm. For the total weighted tardiness criterion, the Tabu Search method yields better solutions in significantly less computing time effort. The Tabu Search method requires about $3$ minutes on average to obtain the best solution, whereas the solution built by LA(MB) needs on average more than $10$ minutes. For the total tardiness and the total weighted completion time criteria, the Tabu Search method yields slightly better solutions than LA(MB), with an equivalent computing effort. Finally, for the total completion time criterion, the Tabu Search method yields slightly less good solutions than LA(MB) with equivalent computing effort.

# 6   Conclusion

In this paper we have described new original dominance rules for $1|r_i|\sum w_i T_i$ and its special cases. We have presented several powerful algorithms based on these dominance rules, which improve well-known heuristic algorithms from the literature. Whereas our Tabu Search method can be significantly enhanced by techniques from the literature specific to the Tabu Search, we have also shown that several techniques based on our dominance rules can greatly improve meta-heuristics such as Tabu Search. Experimental results show the effectiveness of all our algorithms. The approaches we propose improve and outperform the best-known heuristics (algorithms EST, HP, IT, GL and LA used with no improving algorithm) from the literature for $1|r_i|\sum w_i T_i$ and its special cases.

# References

[1] M.S. Akturk and D. Ozdemir. An exact approach to minimizing total weighted tardiness with release dates. *IIE Transactions*, 32:1091–1101, 2000.

[2] M.S. Akturk and D. Ozdemir. A new dominance rule to minimize total weighted tardiness with unequal release dates. *European Journal of Operational Research*, 135:394–412, 2001.

[3] K.R. Baker. *Introduction to Sequencing and Scheduling*. John Wiley and Sons, 1974.

[4] Ph. Baptiste, J. Carlier, and A. Jouglet. A branch-and-bound procedure to minimize total tardiness on one machine with arbitrary release dates. *European Journal of Operational Research*, 158(3):595–608, 2004.

[5] H. Belouadah, M.E. Posner, and C.N. Potts. Scheduling with release dates on a single machine to minimize total weighted completion time. *Discrete Applied Mathematics*, 36:213–231, 1992.

[6] S. Chand, R. Traub, and R. Uzsoy. An iterative heuristic for the single-machine dynamic total completion time scheduling problem. *Computers and Operations Research*, 23:641–651, 1996.

[7] S. Chand, R. Traub, and R. Uzsoy. Rolling horizon procedures for the single machine deterministic total completion time scheduling problem with release dates. *Annals of Operations Research*, 70:115–125, 1997.

[8] C. Chu. A branch and bound algorithm to minimize total flow time with unequal release dates. *Naval Research Logistics*, 39:859–875, 1991.

[9] C. Chu. A branch and bound algorithm to minimize total tardiness with different release dates. *Naval Research Logistics*, 39:265–283, 1992.

[10] C. Chu. Efficient heuristics to minimize total flow time with release dates. *Operations Research Letters*, 12:321–330, 1992.

[11] C. Chu and M.C. Portmann. Some new efficient methods to solve the $n|1|r_i|\sum T_i$ scheduling problem. *European Journal of Operational Research*, 58:404–413, 1991.

[12] R.K. Congram, C.N. Potts, and S.L. Van De Velde. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14:52–67, 2002.

[13] R.W. Conway, W.C. Maxwell, and L.W. Miller. *Theory of scheduling*. Addison Wesley, Reading, MA, 1967.

[14] F. Della Croce and V. T'kindt. A recovering beam search algorithm for the one-machine dynamic total completion time scheduling problem. *Journal of the Operational Research Society*, 53:1275–1280, 2002.

[15] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

[16] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1998.

[17] A.M.A Hariri and C.N. Potts. An algorithm for single machine sequencing with release dates to minimize total weighted completion time. *Discrete Applied Mathematics*, 5:99–109, 1983.

[18] A. Jouglet, Ph. Baptiste, and J. Carlier. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 13: Branch and Bound Algorithms for Total Weighted Tardiness. CRC Press, ed. Joseph Leung, 2004.

[19] J.J. Kanet and Z. Zhou. A decision theory approach to priority dispatching for job shop scheduling. *Production and Operations Management*, 2(1):2–14, 1993.

[20] T.E. Morton and P. Ramnath. *Intelligent Scheduling System*, chapter Guided forward search in tardiness scheduling of large one machine problems. Kluwer Academic Publishers, Hingham, MA, 1995.

[21] A.H.G. Rinnooy Kan. *Machine sequencing problem: classification, complexity and computation*. Nijhoff. The Hague, 1976.

[22] W.E. Smith. Various optimizers for single stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[23] A.P.J. Vepsalainen and T.E. Morton. Priority rules for job shops with weighted tardiness costs. *Management Science*, 39(5):626–632, 1993.