Single-machine scheduling with no idle time and release dates to minimize a regular criterion

Antoine Jouglet

Noname manuscript No.
(will be inserted by the editor)

Single-machine scheduling with no idle time and release dates to minimize a regular criterion

Antoine Jouglet

Abstract We address the one-machine scheduling problem with release dates, in which the machine is subject to the non-idling constraint, *i.e.* no intermediate idle time is permitted between the jobs processed by the machine. The objective is to minimize a regular objective function. We describe a constraint programming approach for solving this type of problem exactly. Some necessary and/or sufficient conditions for obtaining non-idling semi-active, active and optimal schedules are described. We propose some propagation rules based on these properties. As an application, we apply the proposed method to the total (weighted) completion time problem, and we provide some experimental results to illustrate its effectiveness.

Keywords: non-idling scheduling, single machine, release dates, regular criteria, constraint programming.

1 Introduction

We consider the situation where a set $N = \{1, \ldots, n\}$ of n jobs is to be processed by a single machine subject to the non-idling constraint where no intermediate idle time is permitted between the operations processed by the machine (10). This constraint is of great interest in industrial contexts where it is better to process jobs later while ensuring a full machine utilization, rather than stopping and restarting the machine.

Associated with each job i are a release date r_i and a processing time p_i. All data are non-negative integers. A job cannot start before its release date, preemption is not allowed, and only one job at a time can be scheduled on the machine. Let Φ be a regular criterion, *i.e.* an objective function which is nondecreasing with respect to all completion times of jobs (1, 13), *e.g.* the makespan C_{max} or the total completion time $\sum C_i$. The problem is to find a feasible schedule minimizing Φ.

In its classical version without the non-idling constraint, the single-machine problem has been intensively investigated. While some problems such as $1|r_i|C_{max}$ are polynomially solvable, most of them, including $1|r_i|\sum(w_i)T_i$ and $1|r_i|\sum(w_i)C_i$, are known to be NP-hard in the strong sense (22, 25). For these problems various enumeration methods based on dominance properties and lower bounds have been described, *e.g.* (3, 11, 12, 19).

In contrast to the well-known no-wait constraint in shop scheduling, where no idle time is allowed between the successive operations comprising a particular job, the non-idling machine constraint does not seem to have been widely studied in the literature. One of the first explorations of the subject was by Wolf (30) who described the "Reduce-To-The-Opt" algorithm which allows him to solve optimization scheduling problems in which jobs are allowed to take their starts in given sets of possible starting times. In this work, the non-idling machine constraint is called "continuous task scheduling". This algorithm is notably applied to surgery scheduling problems in which different surgeries in operation rooms have to be scheduled contiguously such that some sequence-dependent setup costs are minimized. Later, Valente and Alves (28) developed a branch-and-bound method within the context of the earliness/tardiness single-machine scheduling problem with no unforced idle time. To our knowledge, the most important work is Chrétienne's (10), in which the non-idling machine constraint is introduced. Thus, a machine whose operations must be processed without any intermediate delay is called a **non-idling machine**. By extension, a **non-idling**

Heudiasyc, UMR UTC/CNRS 6599, Université de Technologie de Compiègne, BP 20529, 60205 Compiègne, France
antoine.jouglet@hds.utc.fr

schedule is a schedule for which the non-idling constraint is satisfied on the machine. Chrétienne proposed the notation NI associated with the machine in the 3-field notation. Thus, a problem belonging to the class of problems under consideration is denoted as $1, NI|r_i|\Phi$, where Φ is a regular criterion. Chrétienne studied some aspects of the impact of the non-idling constraint on the complexity of single-machine scheduling problems. Thus, while it is not always true that the non-idling version of an NP-complete single-machine problem is also NP-hard, some problems such as $1, NI|r_i|\sum(w_i)T_i$ and $1, NI|r_i|\sum(w_i)C_i$ are NP-hard in the strong sense. Moreover, Chrétienne looked at some of the issues surrounding non-idling schedules, including how to find the earliest starting time α_{NI} of such a schedule. He derived a sufficient condition such that for any instance of a given classical single-machine problem (without the non-idling constraint), there is an optimal schedule starting at time α_{NI} which is non-idling. A problem with that property is said to have the ENI (Earliest Non-Idling) property and the non-idling variant of this problem can be solved using the same algorithm. In particular, this is the case for single machine problems where preemption is allowed. Thus, $1, NI|r_i, pmtn|\sum C_i$ can be solved in $O(n \log n)$ with the Shortest Remaining Processing Time Rule (26) (see Section 7), by replacing each release r_i by $\max(r_i, \alpha_{NI})$. It should nevertheless be noted that the non-preemptive version of the problem (*i.e.* , $1, NI|r_i|\sum C_i$) does not have the ENI property. More recently, Carlier *et al.* (7) studied the non-idle one-machine sequencing problem in which jobs are subject to release dates and latency durations. This problem, denoted as $1, NI|r_i, q_i|f_{max}$, is strongly NP-Hard (10). Carlier *et al.* (7) described a branch and bound algorithm to solve exactly this problem. They also propose an adaptation of the Jackson's algorithm to polynomially solve the preemptive version $1, NI|pmtn, r_i, q_i|f_{max}$ of the problem.

In this article we describe a constraint programming approach for solving the problem under consideration exactly. In Section 2, we describe our constraint-based branch and bound approach. We next show in Section 3 how the non-idling constraint may be applied. In Section 4 and Section 5, we study respectively the non-idling semi-active and the non-idling active schedules. Properties of such schedules are given, along with propagation rules that depend on these properties. Subsequently, in Section 6, we propose sufficient conditions for eliminating nodes of the search tree which cannot lead to optimal solutions, and in Section 7, by way of an application, we show how the $1, NI|r_i|\sum(w_i)C_i$ problem may be solved. Experimental results are then provided in Section 8 to demonstrate the effectiveness of the method.

2 A constraint-based scheduling method

In this section we propose a constraint-based scheduling approach for solving a problem belonging to the class $1, NI|r_i|\Phi$, where Φ is a regular criterion.

Constraint programming is a programming paradigm aimed at solving combinatorial optimization problems that can be described by a set of variables, a set of possible values for each variable, and a set of constraints between the variables. The set of possible values of a variable x is called the *variable domain* of x and is denoted as $Dom(x)$. A constraint between variables expresses which combinations of values are permitted for the variables. The question is whether there exists an assignment of values to variables such that all constraints are satisfied. The power of the constraint programming method lies mainly in the fact that constraints can be used in an active process termed "constraint propagation" where certain deductions are performed, in order to reduce computational effort. Constraint propagation removes values not belonging to a solution from the domains of variables, deduces new constraints not changing the solution set, and detects inconsistencies.

The principles of constraint programming have been widely applied in the area of scheduling (2). In a constraint-based scheduling model, two variables $start_i$ and end_i representing respectively the starting time and the completion time are associated with each job i. The constraint $start_i + p_i = end_i$ is maintained. The minimum and maximum values of the domain of $start_i$ are respectively denoted as:

- $est_i = \min_{v \in Dom(start_i)} v$: the earliest starting time of job i,
- $lst_i = \max_{v \in Dom(start_i)} v$: the latest starting time of job i.

Similarly, the minimum and maximum values of the domain of end_i are respectively denoted as:

- $eet_i = \min_{v \in Dom(end_i)} v = est_i + p_i$: the earliest completion time of job i,
- $let_i = \max_{v \in Dom(end_i)} v = lst_i + p_i$: the latest completion time of job i.

Thus, each job i has to be executed within the time window $[est_i, let_i)$ whose bounds can be initialized with $est_i = r_i$ and $let_i = \max_{i \in N} r_i + \sum_{i \in N} p_i$ (see Section 4, Corollary 1). Note that the domains of variables $start_i$ and end_i will be reduced all along the search using propagation algorithms. Thus, the bounds of the domains of $start_i$ and end_i (*i.e.* est_i, lst_i, eet_i and let_i) cannot be expressed with formal equations

since they are the results of some decisions taken all along the search. We use edge-finding propagation techniques (2) which are able to adjust the time-windows of jobs according to the disjunctive constraints between jobs and their possible starting times. Note that when the starting times of jobs have been fixed we have, for each job i, $start_i = est_i = lst_i$ and $end_i = eet_i = let_i = start_i + p_i$.

Two additional variables $Start$ and End represent respectively the start and completion times for the entire schedule. $Start = \min_{i \in N} start_i$ and $End = \max_{i \in N} end_i$. The constraint $Start + P = End$ is maintained where $P = \sum_{i \in N} p_i$ is the total processing time of the jobs. The notation $Est = \min_{v \in Dom(Start)} v$, $Eet = \min_{v \in Dom(End)} v$, $Lst = \max_{v \in Dom(Start)} v$ and $Let = \max_{v \in Dom(End)} v$ will be used to denote respectively the lower bounds and the upper bounds of the domain of these variables.

An additional constrained variable $\overline{\Phi}$ represents the objective function to be optimized. For instance, the variable $\overline{\Phi} = \sum_{i=1}^{n} w_i \cdot end_i$ is used for the $1, NI|r_i|\sum w_i C_i$ problem. Arc-B-Consistency (see for instance (23)) is used to propagate this constraint, ensuring that when a schedule has been found the value of $\overline{\Phi}$ is actually equal to the value of the studied criterion of the schedule.

To find an optimal solution we use a branch and bound algorithm solving successive variants of the decision problem. At each iteration we try to improve the best known solution and to this end we add an additional constraint stating that $\overline{\Phi}$ is lower than or equal to the best solution minus 1. Each time, the search resumes at the last-visited node of the previous iteration.

To solve the decision variant of the problem, the same scheme as in (19) is used, and we make use of the Edge-Finding branching scheme (see for instance Carlier (6)). This involves ordering jobs on the machine ("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 or last among the jobs in this set. Consequently, rather than searching for the starting times of jobs, we look for a **sequence** of jobs. This is why this scheme is particularly suitable for the non-idling constraint. In the version of the problem without the non-idling constraint, being given a partial sequence of jobs to be scheduled at the beginning of the schedule, jobs can be scheduled as soon as possible in the order of the sequence without changing the value of an optimal schedule which starts with this sequence (see for instance (1)). Here the starting times of jobs in a partial sequence are not necessarily known, since they are highly dependent on the next decisions to be taken (which would have led to idle times in the classical problem). Nevertheless, the domains of the starting times of these jobs will be adjusted according to the propagation rules described in the following sections. The sequence is built both from the beginning and

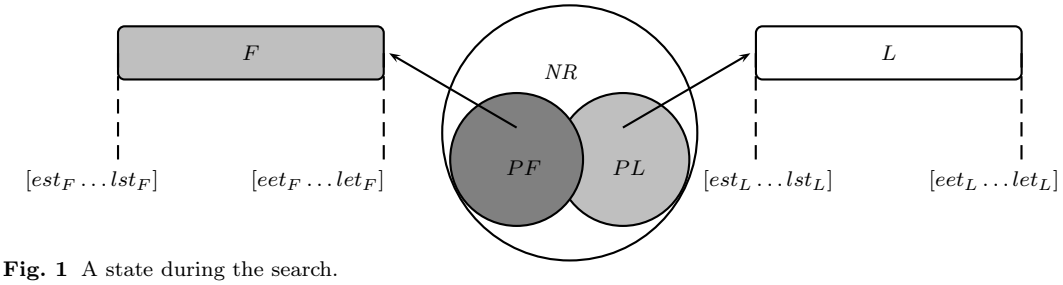

Fig. 1 A state during the search.

from the end of the schedule using a **depth-first strategy**. Throughout the search tree, we dynamically maintain several sets of jobs that represent the current state of the schedule (see Figure 1):

- F (First) is the sequence of jobs sequenced at the beginning,
- L (Last) is the sequence of jobs sequenced at the end,
- NR (Not Ranked) is the set of as yet unranked jobs which are to be sequenced between F and L,
- $PF \subseteq NR$ (Possible First) is the set of jobs which can be ranked immediately after F,
- and $PL \subseteq NR$ (Possible Last) is the set of jobs which can be ranked immediately before L.

By extension, we use the notation est_F, lst_F, eet_F and let_F to designate respectively the earliest start time, the latest start time, the earliest end time and the latest end time for sequence F. If F is empty we have $est_F = eet_F = Est$ and $lst_F = let_F = Lst$, otherwise (if there is at least one job which has been sequenced at the beginning) we have $est_F = \min_{i \in F} est_i = Est$, $lst_F = \min_{i \in F} lst_i = Lst$, $eet_F = \max_{i \in F} eet_i$ and $let_F = \max_{i \in F} let_i$. Similarly, we use the notation est_L, lst_L, eet_L and let_L to designate respectively the earliest start time, the latest start time, the earliest end time and the latest end time for sequence L. If L is empty we have $est_L = eet_L = Eet$ and $lst_L = let_L = Let$, otherwise (if there is at least one job which has been sequenced at the end) we have $est_L = \min_{i \in L} est_i$, $lst_L = \min_{i \in L} lst_i$, $eet_L = \max_{i \in L} eet_i = Eet$ and $let_L = \max_{i \in L} let_i = Let$.

At each node of the search tree, sets PF and PL are initialized with NR and are filtered using techniques described in the next sections. Next, a new node is derived from the current node by the following way. Either a job is selected from PF and ranked immediately after F, or it is selected from PL and ranked immediately before L. Upon backtracking, this job is removed from PF or PL. Of course, if NR is empty then a solution has been found and we can iterate to the next decision problem. If $NR \neq \emptyset$ while PF or PL is empty then a backtrack occurs. Note that if a job i is neither in PF nor in PL, it does not mean that no solution can be obtained from this node. It only means that no solution can be derived from this node in which i is sequenced just after F or just before L.

Each time a job $i \in PF$ is ranked immediately after F then the constraint $start_j + p_j = start_i$ is added, where j is the last job in sequence F if it is not empty. Moreover, each time a job $i \in PL$ is ranked immediately before L, then the constraint $start_i + p_i = start_j$ is added, where j is the first job in sequence L if it is not empty. Adding these constraints allows us to adjust the domains of starting times of ranked jobs according to the domain of starting times for jobs in the same sequence. Thus, relations, $est_F + \sum_{i \in F} = eet_F$, $lst_F + \sum_{i \in F} = let_F$, $est_L + \sum_{i \in L} = eet_L$, $lst_L + \sum_{i \in L} = let_L$ hold. Note that the set of these constraints and the constraint $Start + P = End$ represent the non idling constraint in our approach.

We use the term "partial solution" to designate a state of the variables at a node of the search tree, in which some jobs are already ranked in F or L while some other ones are unranked and belong to NR. Below we shall use the notation $start_i(S)$, $est_i(S)$, $lst_i(S)$, ..., $Lst(S)$, $Let(S)$, ..., $F(S)$, $L(S)$, $PF(S)$, ...to represent the state of the variables in a partial solution S. Thus, note that a partial solution S represents all the solutions which can be obtained with a fixed list $F(S)$ of first tasks and a fixed list $L(S)$ of last tasks. **From now on, only partial solutions as described above will be considered in the remainder of the paper**.

3 Propagating the non-idling constraint

The $Start + P = End$ constraint (see the previous section) is sufficient to assure that once all job starting times have been fixed, the obtained schedule is non-idling. In this section we describe an algorithm which propagates the non-idling constraint by adjusting the domain of variables $Start$ and End of a partial schedule according to the domain of variables associated with job start and completion times.

From (10), given a sequence $\sigma = (\sigma_1, \ldots, \sigma_n)$, the earliest non-idling feasible schedule respecting this sequence of jobs starts at time

$$\alpha_{NI}(\sigma) = \max\left(0, \max_{k \in N}(r_{\sigma_k} - \sum_{q=1}^{k-1} p_{\sigma_q})\right).$$

Moreover, the value $\alpha_{NI} = \alpha_{NI}(\sigma_r)$ obtained from the sequence σ_r in which the jobs are sorted in nondecreasing order of release dates, is the earliest starting time of a feasible non-idling schedule (10).

Noting that $\max_{k \in N}(r_{\sigma_k} - \sum_{q=1}^{k-1} p_{\sigma_q}) = \max_{k \in N}(r_{\sigma_k} - \sum_{q=1}^{k-1} p_{\sigma_q} + \sum_{q=1}^{k-1} p_{\sigma_q} + \sum_{q=k}^{n} p_{\sigma_q} - P) = \max_{k \in N}(r_{\sigma_k} + \sum_{q=k}^{n} p_{\sigma_q}) - P$ and that a sequence of jobs cannot start before $\min_{k \in N} r_k$, we can equivalently express $\alpha_{NI}(\sigma)$ as follows:

$$\alpha_{NI}(\sigma) = \max\left(\min_{k \in N} r_k, \max_{k \in N}(r_{\sigma_k} + \sum_{q=k}^{n} p_{\sigma_q}) - P\right). \quad (1)$$

Symmetrically, this result can be used to compute the latest completion time of a feasible non-idling schedule where a deadline $\overline{d}_i$ is associated with each job i (*i.e.* each job i has to be terminated before $\overline{d}_i$) and where the jobs are released at time 0 (*i.e.* $\forall i, r_i = 0$). Given a sequence $\sigma = (\sigma_1, \ldots, \sigma_n)$, the latest non-idling feasible schedule respecting this sequence of jobs ends at time

$$\omega_{NI}(\sigma) = \min\left(\max_{k \in N} \overline{d}_k, \min_{k \in N}(\overline{d}_{\sigma_k} - \sum_{q=1}^{k} p_{\sigma_q}) + P\right). \quad (2)$$

Moreover, the value $\omega_{NI} = \omega_{NI}(\sigma)$ obtained from the sequence σ in which the jobs are sorted in nondecreasing order of deadlines is the latest starting time of a feasible non-idling schedule.

These results can be used in Algorithm 1 to adjust the bounds of the domains of variables $Start(S)$ and $End(S)$ (*i.e.* $Est(S)$, $Lst(S)$, $Eet(S)$ and $Let(S)$) of a given partial solution S (as defined in Section 2). Equations 1 and 2 are used taking into account the domain of variables $start_i(S)$ and $end_i(S)$ of each job i

in the partial solution S. Thus, jobs being sorted in nondecreasing order of earliest start times, the release date r_i of a job i in Equation 1 is replaced by the earliest start time $est_i(S)$. It allows to compute an earliest start time for the partial solution S. Similarly, jobs being sorted in nondecreasing order of latest end times, the deadline $\overline{d}_i$ of a job i in Equation 2 is replaced by the latest end time $let_i(S)$. Thus, it allows to compute a latest end time for the partial solution S. Since the sort of jobs in nondecreasing order of earliest start times and next in nondecreasing order of latest end times can be done in $O(n \log n)$, Algorithm 1 allows us to adjust in $O(n \log n)$ the bounds of domains of variables *Start* and *End*.

Algorithm 1: Propagating the non-idling constraint on *Start* and *End* variables

Data: A partial solution S (*cf.* Figure 1)
$\sigma \leftarrow$ the sequence of jobs sorted in nondecreasing order of $est_i(S)$;
$t \leftarrow Est(S)$;
for $i \leftarrow 1$ *to* n **do** $t \leftarrow \max(t, est_{\sigma(i)}(S)) + p_{\sigma(i)}$;
$Eet(S) \leftarrow \max(Eet(S), t)$ and $Est(S) \leftarrow \max(Est(S), t - P)$;
$\sigma \leftarrow$ the sequence of jobs sorted nondecreasing order of $let_i(S)$;
$t \leftarrow Let(S)$;
for $i \leftarrow n$ *to* 1 **do** $t \leftarrow \min(t, let_{\sigma(i)}(S)) - p_{\sigma(i)}$;
$Lst(S) \leftarrow \min(Lst(S), t)$ and $Let(S) \leftarrow \min(Let(S), t + P)$;

Unfortunately, since we are using release dates (earliest starting times) and deadlines (latest finishing times) simultaneously, Algorithm 1 is not able to perform all possible deductions. Take, for example, a problem with 3 jobs where $p_1 = 3$, $est_1 = 0$, $let_1 = 15$, $p_2 = 3$, $est_2 = 1$, $let_2 = 15$, $p_3 = 4$, $est_3 = 5$, $let_3 = 9$. Algorithm 1 computes $Est = 0$ and $Let = 15$, but we observe that no non-idling feasible schedule can start before 2. This fact cannot be deduced using Edge-Finding techniques, since $est_1 = 0$ and $est_2 = 1$ are possible starting times in feasible schedules where the non-idling constraint is not imposed.

After experimental results, we have noted that in practice this case is unlikely to be encountered often. We may use techniques based on the solution of subset-sum problems to refine the adjustments on *Est* or *Let*. In the previous example it was possible to deduce that $Est = 2$. However, after performing experimental tests, we have noted that in practice such deductions do not improve the performance of the search method.

4 Non-idling semi-active schedules

In this section we show that the notion of a semi-active schedule can be applied to non-idling scheduling. We present a necessary and sufficient condition for a non-idling schedule to be semi-active, which allows us to show that the set of semi-active schedules is dominant for the regular criteria, *i.e.* that there exists at least one optimal schedule which is semi-active (1). We also establish some properties for such schedules. We then describe an algorithm for propagating the non-idling semi-active constraint.

A semi-active schedule is defined as a schedule in which no job can be scheduled earlier without changing the sequence of execution of jobs on the machine or violating the constraints (5). Applied to non-idling problems we obtain the following definition:

Definition 1 A non-idling feasible schedule is said to be semi-active if no job can be scheduled earlier without either changing the sequence of execution of jobs on the machine or violating a model constraint (including the non-idling constraint).

However, this property is not the same as in the case of semi-active schedules in the classical problems. For example, Figure 2 shows two schedules S and S' for the same sequence of jobs. S is semi-active for the

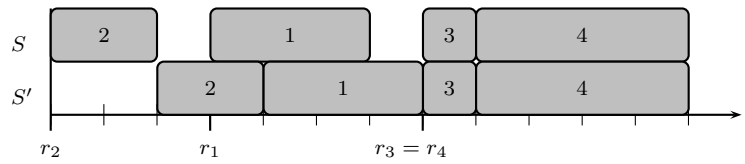

Fig. 2 Only S' is a non-idling semi-active schedule.

classical problem, while S' is not. However, S' is a non-idling semi-active schedule while S is not.

We now give a necessary and sufficient condition for a non-idling schedule to be semi-active relying on the fact that if no job starts at its release date in a non-idling schedule, the jobs can be scheduled earlier without changing the sequence of execution of jobs.

Theorem 1 *A non-idling schedule S is semi-active if and only if there is at least one job which starts at its release date, i.e.* $\min_{i \in N} (start_i(S) - r_i) = 0$.

An upper bound of the makespan in a non-idling semi-active schedule is provided in the following corollary:

Corollary 1 *The latest start and completion times for a non-idling semi-active schedule are respectively* $\max_{i \in N} r_i$ *and* $\max_{i \in N} r_i + \sum_{i \in N} p_i$, *and these bounds are tight.*

Proof If a schedule S starts strictly after $\max_{i \in N} r_i$, *then no job can start at its release date and S is not semi-active. Moreover, any semi-active schedule whose first job has the latest release date starts at* $\max_{i \in N} r_i$. *If all other jobs are scheduled as soon as possible after this first job, then whatever the sequence the obtained schedule is semi-active and ends at* $\max_{i \in N} r_i + \sum_{i \in N} p_i$.

It is well known that the subset of semi-active schedules is dominant (*i.e.* there exists at least one optimal non-idling schedule which is semi-active) for problems with a regular criterion (1), and the following proposition can therefore be obviously derived:

Proposition 1 *The set of non-idling semi-active schedules is dominant (i.e. there exists at least one optimal non-idling schedule which is semi-active) for non-idling problems where a regular criterion is to be minimized.*

Consequently, only non-idling semi-active solutions need to be considered, and the following proposition provides a necessary condition that a partial solution S starting at its current latest starting time $Lst(S)$ has to fulfill to yield a non-idling semi-active schedule. In this proposition, for each job i the value $\delta_i(S)$ is a lower bound of the distance between its release date r_i and its starting time in a schedule obtained from S starting at $Lst(S)$.

Proposition 2 *Given a partial solution S, let $\delta_i(S)$ be equal to:*

- $\delta_i(S) = lst_i(S) - r_i$ *if* $i \in F(S) \cup L(S)$, *i.e. if i has already been ranked first or last;*
- $\delta_i(S) = \max(let_F(S), est_i(S)) - r_i$ *if* $i \in PF(S)$;
- $\delta_i(S) = \max(let_F(S) + min_{j \in PF(S)}(p_j), est_i(S)) - r_i$ *if* $i \in NR(S) \backslash PF(S)$.

If there exists a non-idling semi-active schedule obtained from S and which starts at $Lst(S)$, then we must have $min_{i \in N} \delta_i(S) = 0$.

Proof Consider any schedule S' obtained from S starting at $Lst(S)$. Then every job $i \in F(S) \cup L(S)$ starts exactly at $start_i(S') = lst_i(S)$ in S'. A job $i \in PF(S)$ starts not earlier than $\max(let_F(S), est_i)$. *In S', since a job $j \in PF(S)$ is necessarily scheduled before a job $i \in NR(S) \backslash PF(S)$, i starts not earlier than* $\max(let_F(S) + min_{j \in PF(S)} p_j, est_i(S))$. *Consequently, for each job i the value $\delta_i(S)$ is a lower bound of the distance between its release date r_i and its start time $start_i(S')$. Thus, if $min_{i \in N} \delta_i(S) > 0$, then S' is not semi-active.*

This proposition yields Algorithm 2, which runs in $O(n)$ time since value δ_i has to be computed for each job i. It enables us to perform adjustments on a partial solution S. If no job can start at its release date, then a backtrack occurs, since S cannot lead to a semi-active schedule. If only one job can start at its release date, then it is obliged to do so (see Theorem 1). Moreover, the upper bounds of the domains of variables *Start* and *End* (*i.e. Lst* and *Let*) are adjusted in order to satisfy Proposition 2.

Algorithm 2: Propagating the non-idling semi-active constraint

Data: a partial solution S (*cf.* Figure 1)
$X \leftarrow \{i / est_i(S) = r_i\}$;
if $X = \emptyset$ **then** a backtrack is triggered;
if $X = \{i\}$ **then** $start_i(S) \leftarrow r_i$;
$Lst(S) \leftarrow Lst(S) - min_{i \in N} \delta_i(S)$;
$Let(S) \leftarrow Lst(S) + P$;

5 Non-idling active schedules

In this section we show that the notion of an active schedule can be applied to non-idling scheduling in the same way as for semi-active schedules. We establish that the set of non-idling active schedules is dominant (*i.e.* there is at least one optimal schedule which is non-idling active) for the regular criterion, and present a necessary condition for a non-idling schedule to be active. Using this condition we put forward three propositions for adjusting the variables' domains of a partial schedule, and then describe an algorithm based on these propositions for propagating the non-idling active constraint.

Like in the case of semi-active schedules, the notion of non-idling active schedules can be defined as follows:

Definition 2 A non-idling feasible schedule is said to be active if no job can be completed earlier without either delaying another job or violating a model constraint (including the non-idling constraint).

Nevertheless, once again this property is not the same as for the active schedules of the classical problem.

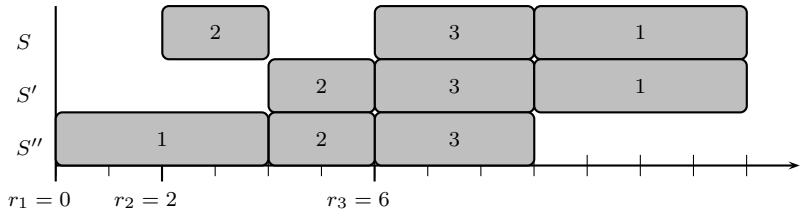

Fig. 3 S is an active schedule, S'' is a non-idling active schedule.

For example, in the instance of Figure 3 we have $r_1 = 0$, $p_1 = 4$, $r_2 = 2$ $p_2 = 2$, $r_3 = 6$, $p_3 = 3$. Schedule S is active for the $1|r_i|\Phi$ problem while it is not feasible for $1, NI|r_i|\Phi$ since it is not idling. Schedule S', with the same sequence as S, is non-active for both $1|r_i|\Phi$ and $1, NI|r_i|\Phi$ since it is possible to schedule job 1 with $start_1 = 0$ without delaying jobs 2 and 3, as shown in schedule S'', which, however, is an active schedule both for $1|r_i|\Phi$ and $1, NI|r_i|\Phi$.

It is hard to find a direct relation between sequences yielding active schedules for the classical problem and those yielding non-idling active schedules. Indeed, as shown in the previous example, a sequence of jobs yielding an active schedule for the classical problem does not necessarily yield a non-idling active schedule. Moreover, a sequence of jobs yielding a non-idling active schedule does not necessarily yield an active schedule for the classical problem. There is, however, an obvious relation between non-idling semi-active schedules and non-idling active schedules:

Proposition 3 *A non-idling active schedule is a non-idling semi-active schedule.*

Proof Consider a schedule S which is not semi-active. A job may therefore be scheduled earlier without changing the sequence of jobs and without violating the non-idling constraint. Consequently, S is not active.

It is well known that the subset of active schedules is dominant (*i.e.* there is at least one optimal schedule which is non-idling active) for problems with a regular criterion (1), and the following propositions may therefore be derived:

Proposition 4 *The set of non-idling active schedules is dominant (i.e. there is at least one optimal schedule which is non-idling active) for non-idling problems where a regular criterion has to be minimized.*

Proof Consider a non-idling schedule S which minimizes a regular objective function. If S is not active, then there exists a job i which can be scheduled earlier without violating a model constraint and without delaying another job. This yields a non-idling schedule S' with a cost not greater than that of S, and which is therefore optimal. The process is repeated until no such a job i exists. Thus an optimal non-idling active schedule has been obtained.

The following theorem provides a necessary condition for a schedule S to be a non-idling active schedule. It relies on the fact that if the condition is not satisfied for a schedule S, a job i may be inserted earlier without delaying the other jobs. It covers all insertions of this kind through considering the longest possible one.

Theorem 2 *If S is a non-idling active schedule then:*

$$\forall i \in N \min_{\substack{\{j, j\neq i / start_j(S) < r_i + p_i \\ \vee start_j(S) > start_i(S)\}}} \left(start_j(S) - r_j\right) < p_i \quad (3)$$

Proof Consider a non-idling active schedule S where Condition 3 is not satisfied for a job i. Since the schedule is active, it is also semi-active, and a job cannot be scheduled earlier simply by scheduling the entire schedule earlier. Let $A = \{j / start_j(S) > start_i(S)\}$ be the set of jobs which are scheduled after i, let $B = \{j / start_j(S) < r_i + p_i\}$ be the set of jobs starting before $r_i + p_i$, and let $C = \{j / r_i + p_i \leq start_j(S) < start_i(S)\}$ be the set of jobs which are scheduled between jobs belonging to B and job i (see Figure 4). If

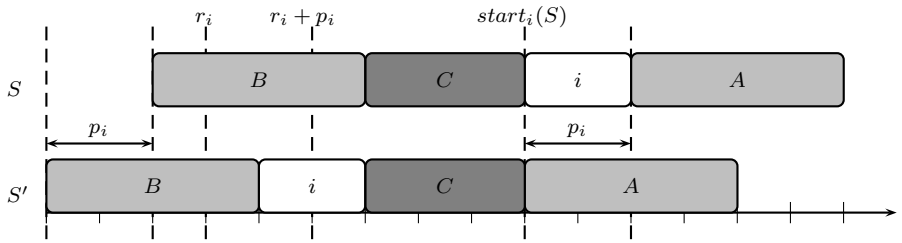

Fig. 4 S is a non-idling active schedule.

Condition 3 is not satisfied, this means that $\min_{j\in A}\left(start_j(S) - r_j\right) \geq p_i$. *Thus, job i can be removed from the schedule and the jobs belonging to A can be scheduled p_i units of time earlier since they are scheduled at least p_i units of time from their release dates. The non-idling constraint is thus restored. Jobs belonging to set B can be also scheduled p_i units of time earlier since* $\min_{\{j\in B\}}\left(start_j(S) - r_j\right) \geq p_i$. *Job i can then be inserted just after B without delaying subsequent jobs, giving us a non-idling schedule S' in which job i has been scheduled earlier without delaying any other job. This contradicts the fact that S is active. Note that the starting times of jobs belonging to C do not change during the move of i. Note also that if C is empty, it means that the whole schedule can be simply brought forward from at least p_i units of time contradicting the fact that S is semi-active and active. Finally, note that this condition covers all possible insertions of i before a job in C, since C has been chosen to be maximal.*

Note that this condition is not sufficient, since a non-idling non-active schedule satisfying Condition 3 may exist of Theorem 2. For example, in the instance of Figure 5, we have $r_1 = 1$, $r_2 = 0$, $r_3 = 7$, $r_4 = 4$,

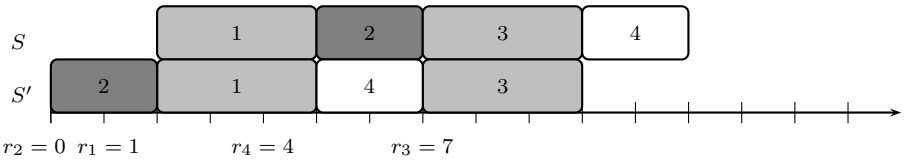

Fig. 5 A non-idling non-active schedule S not satisfying Condition 3

$p_1 = p_3 = 3$ and $p_2 = p_4 = 2$. Schedule S satisfies Condition 3 while it is non-idling non-active since jobs 2 and 4 can be scheduled earlier without delaying another job as shown in schedule S'.

Below we provide some sufficient conditions under which ranking a job in the first or the last position, or scheduling a job after a certain date according to a partial solution S, cannot yield an active schedule. These conditions are all based on Condition 3 of Theorem 2. From these conditions we then derive an algorithm for propagating the non-idling active constraint.

In the following proposition we establish a sufficient condition ensuring that ranking a job i immediately after $F(S)$ yields a partial solution in which i can be scheduled earlier, regardless of how subsequent jobs are scheduled. We do this by showing that it leads to a partial solution that fails to satisfy Condition 3 of Theorem 2.

Proposition 5 *Given a partial solution S, let i be a job belonging to $NR(S)$ such that $eet_i(S) \geq \max_{j\in NR(S)} r_j$ and let $\delta = est_i(S) - eet_F(S)$. If there exists a job $x \in NR(S)\backslash\{i\}$ such that:*

$$r_x + p_x \leq est_i(S) \wedge \min_{j\in L(S)\cup\{z\in F(S) / est_z(S)+\delta < r_x + p_x\}} \left(est_j(S) + \delta - r_j\right) \geq p_x$$

then ranking i immediately after $F(S)$ cannot lead to a non-idling active schedule.

Proof Consider such a partial solution S and such jobs i and x. Let S' be a non-idling schedule obtained from S by ranking i immediately after $F(S)$ and by completing the schedule with any sequence of jobs belonging to $NR(S)\backslash\{i\}$ between i and $L(S)$ (see Figure 6). Let z be the last job in sequence $F(S)$. Since job i

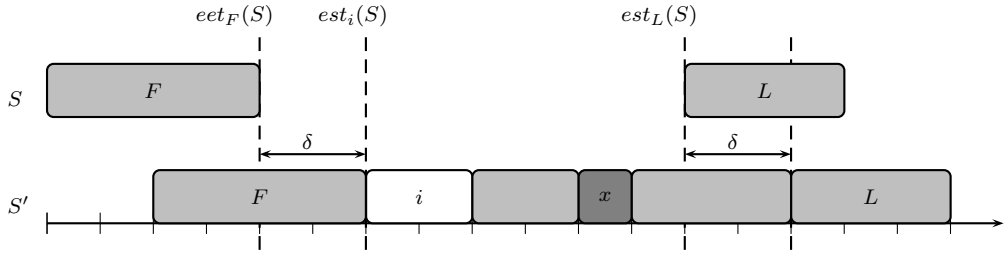

Fig. 6 Scheduling i immediately after $F(S)$ cannot lead to a non-idling active schedule.

is scheduled immediately after job z, then $end_z(S') \geq est_i(S)$, otherwise the non-idling constraint is violated in S'. This means that all jobs in $F(S)$ and $L(S)$ are scheduled at least $\delta = est_i(S) - eet_F(S)$ units of time later in S' than their earliest starting times in S. Consequently, we have $\forall j \in F(S) \cup L(S), start_j(S') \geq est_j(S) + \delta$. Thus, all jobs belonging to $F(S)$ such that $est_j(S) + \delta < r_x + p_x$ satisfy $start_j(S') - r_j \geq est_j(S) + \delta - r_j \geq p_x$. Job x is scheduled after i in S' and we have $\max_{j \in NR(S)} r_j \leq eet_i(S) \leq end_i(S') \leq start_x(S')$. Moreover, all jobs j belonging to $L(S)$ are such that $start_j(S') \geq est_j(S) + \delta$. Consequently, all jobs j scheduled after x, including jobs in $L(S)$, are such that $start_j(S') - r_j \geq p_x$. Since S' does not fulfill Condition 3 of Theorem 2, then S' is not active.

In the following proposition we establish a sufficient condition ensuring that ranking a job i immediately before $L(S)$ yields a partial solution in which i can be scheduled earlier, regardless of how subsequent jobs are scheduled. To do this, once again we show that it leads to a partial solution which does not satisfy Condition 3 of Theorem 2.

Proposition 6 *Given a partial solution S, let $i \in NR(S)$ such that $\min_{\{j \in NR(S)\backslash\{i\}\}} est_j(S) \geq r_i + p_i$ and let $\delta = \min_{j \in NR(S)\backslash\{i\}} est_j(S) - eet_F(S)$. If $\min_{j \in L(S) \cup \{z \in F(S) / est_z(S) + \delta < r_i + p_i\}} (est_j(S) + \delta - r_j) \geq p_i$ then ranking i immediately before $L(S)$ cannot lead to a non-idling active schedule.*

Proof Consider such a partial solution S and such a job i. Let S' be a schedule obtained from S by ranking i immediately before $L(S)$ and by completing the schedule with any sequence of jobs of $NR(S)\backslash\{i\}$ between $F(S)$ and i (see Figure 7). Note that $\min_{\{j \in NR(S)\backslash\{i\}\}} est_j(S) \geq r_i + p_i$ implies that every job belonging

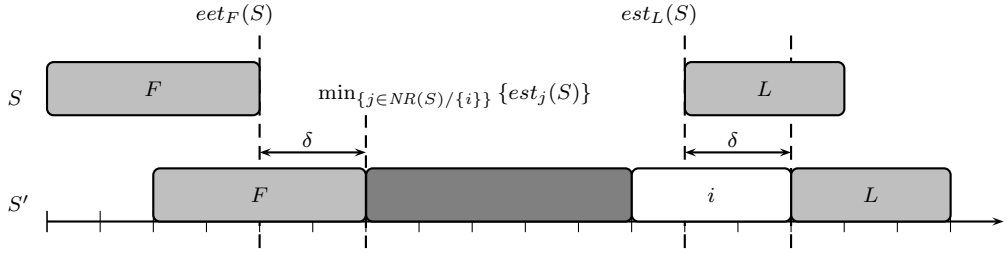

Fig. 7 Scheduling i immediately before $L(S)$ cannot lead to a non-idling active schedule.

to $NR(S)\backslash\{i\}$ start not earlier than $r_i + p_i$ in S' and that a job j which can satisfy $start_j(S') < r_i + p_i$ is necessarily sequenced in $F(S)$. Let z be the last job in $F(S)$. Since i is not scheduled immediately after $F(S)$ and since S' is a non-idling schedule, then $end_z(S') \geq \min_{\{j \in NR(S)\backslash\{i\}\}} est_j(S)$. This means that all jobs in $F(S)$ and $L(S)$ are scheduled at least $\delta = \min_{\{j \in NR(S)\backslash\{i\}\}} est_j(S) - eet_F(S)$ units of time later in S' than their earliest starting times in S. Consequently, we have $\forall j \in F(S) \cup L(S), start_j(S') \geq est_j(S) + \delta$. Thus, all jobs belonging to $F(S)$ such that $est_j(S) + \delta < r_i + p_i$ and all jobs scheduled after i, i.e. belonging to $L(S)$, satisfy $start_j(S') - r_j \geq est_j(S) + \delta - r_j \geq p_i$. Since S' does not fulfill Condition 3 of Theorem 2, then S' is not active.

In the following proposition we establish a sufficient condition ensuring that scheduling a job i after a certain date always yields a schedule in which i can be scheduled earlier. To this end we consider a schedule in which i is scheduled after a certain date and we show that Condition 3 of Theorem 2 is not fulfilled.

Proposition 7 *Given a partial solution S, each job $i \in NR(S)$ ends strictly before*

$$\overline{d}_i(S) = p_i + \max \begin{cases} est_L(S) - \min_{j \in L(S) \cup \{z \in F(S)/est_z(S) < r_i + p_i\}} (est_j(S) - r_j) \\ \sum_{z \in NR(S)} p_z + \max_{\{j \in NR(S) \setminus \{i\}/est_j(S) - r_j < p_i\}} r_j \end{cases}$$

in any non-idling active schedule derived from S.

Proof *Consider a partial solution S and a job $i \in NR(S)$. Let S' be a schedule obtained from S by completing the schedule with any sequence of jobs of $NR(S)$ between $F(S)$ and $L(S)$. Suppose that i ends at time $t \geq \overline{d}_i(S)$. In S',sequence $L(S)$ cannot start earlier than t. Consequently, any job j belonging to $F(S) \cup L(S)$ is scheduled in S' at time $start_j(S') \geq est_j(S) + t - est_L(S)$. Indeed, all jobs in $F(S)$ and $L(S)$ are scheduled at least $t - est_L(S)$ units of time later in S' than their earliest starting times in S. Therefore, since $t \geq \overline{d}_i(S)$, for all jobs j belonging to $L(S) \cup \{z \in F(S)/est_z(S) < r_i + p_i\}$, we have $start_j(S') \geq est_j(S) + p_i + est_L(S) - est_j(S) + r_j - est_L(S) = p_i + r_j$ and thus $start_j(S') - r_j \geq p_i$.*

In S', a job j belonging to $NR(S) \setminus \{i\}$ starts at $start_j(S') \geq \max\{est_j(S), t - \sum_{z \in NR(S)} p_z\}$ considering the extreme case where j is scheduled immediately after $F(S)$ and job i is scheduled immediately before $L(S)$. For all those jobs where $est_j(S) - r_j \geq p_i$, we have $start_j(S') - r_j \geq p_i$. For all the others, i.e. $\{j \in NR(S) \setminus \{i\}/est_j(S) - r_j < p_i\}$, we have $t \geq \overline{d}_i(S) \geq p_i + r_j + \sum_{z \in NR(S)} p_z$ implying that $start_j(S') \geq p_i + r_j + \sum_{z \in NR(S)} p_z - \sum_{z \in NR(S)} p_z = p_i + r_j$ and $start_j(S') - r_j \geq p_i$.

Consequently, having $t \geq \overline{d}_i(S)$ implies that for all jobs scheduled before $r_i + p_i$ or after i in S', we have $start_j(S') - r_j \geq p_i$. Consequently S' does not fulfill Condition 3 of Theorem 2 and is therefore not active.

Since the set of non-idling active schedules is dominant, only non-idling active solutions need to be considered. Throughout the search, sets PF and PL can thus be filtered using Proposition 5 and Proposition 6. Moreover, Proposition 7 can be used to adjust the latest end times of unranked jobs. All these deductions can be performed by Algorithm 3, based on these propositions. For each unranked job i, computing $\overline{d}_i(S)$

Algorithm 3: Propagating the non-idling active constraint

Data: A partial solution S (*cf.* Figure 1)
foreach $i \in NR(S)$ **do**
  $let_i(S) \leftarrow \min(let_i(S), \overline{d}_i(S) - 1)$;
  if $i \in PF(S)$ **then**
    $\delta \leftarrow est_i(S) - eet_F(S)$;
    if $\exists x \in NR(S) \setminus \{i\}/r_x + p_x \leq est_i(S) \wedge \min_{j \in L(S) \cup \{z \in F(S)/est_z(S) + \delta < r_x + p_x\}} (est_j(S) + \delta - r_j) \geq p_x$
    then $PF(S) \leftarrow PF(S) \setminus \{i\}$;
  if $i \in PL(S)$ **then**
    $\delta \leftarrow \min_{j \in NR(S) \setminus \{i\}} (est_j(S)) - eet_F(S)$;
    if $\min_{j \in L(S) \cup \{z \in F(S)/est_z(S) + \delta < r_i + p_i\}} (est_j(S) + \delta - r_j) \geq p_i$ **then** $PL(S) \leftarrow PL(S) \setminus \{i\}$;

can be done in $O(n)$ time, searching a job x satisfying Proposition 5 can be done in $O(n \cdot |NR(S)|)$ time and verifying if the condition of Proposition 6 is satisfied can be done in $O(n)$. Therefore, Algorithm 3 runs in $O(n \cdot |NR(S)|^2)$ time since there are $|NR(S)|$ unranked jobs to consider.

6 Eliminating non-optimal schedules

In this section, only problems for which Φ is additively separable into nondecreasing functions of individual end times are considered. It will be remarked that this is the case for most of the regular criteria used in scheduling (see for example (5)).

Jouglet *et al.* (19) introduced the notion of "better" sequences for the $1|r_i|\sum(w_i)T_i$ and $1|r_i|\sum(w_i)C_i$ problems, which enabled them to compare two partial sequences σ and σ' in the same set of jobs (*i.e.* , σ is a permutation of σ'). Informally speaking, a sequence σ' is said to be "better" than a sequence σ if replacing σ by σ' in any feasible schedule which starts with the sequence σ, we obtain a schedule with

a lower cost. Following the same idea, in this section we put forward some sufficient conditions whereby a partial solution will not yield an optimal solution. We then propose techniques for filtering the sets $PF(S)$ and $PL(S)$ of a partial solution S according to these sufficient conditions. Note that, thanks to the non-idling constraint, such dominance rule can consider any subsequence of jobs, not necessarily at the beginning of the whole sequence, unlike that in (19). In particular, such a dominance rule can be applied to $L(S)$.

From now on $\alpha_{NI}(\sigma)$ will refer to the earliest starting time of a non-idling schedule following the sequence σ (see Section 3), and $\Phi_\sigma(t)$ will denote the cost of the schedule where jobs belonging to sequence σ are scheduled as early as possible after t in the order of the sequence. Note that only for $t \geq \alpha_{NI}(\sigma)$, the schedule obtained for the jobs in σ is non-idling. Moreover, given two sequences of jobs σ_1 and σ_2, we use the notation $\sigma_1 \circ \sigma_2$ to represent the sequence resulting from the concatenation of sequences σ_1 and σ_2.

We rely on the following idea. Consider a non-idling schedule S implementing a sequence which is separated into three complementary subsequences $\rho \circ \sigma \circ \tau$ (see Figure 8). If we can find a permutation

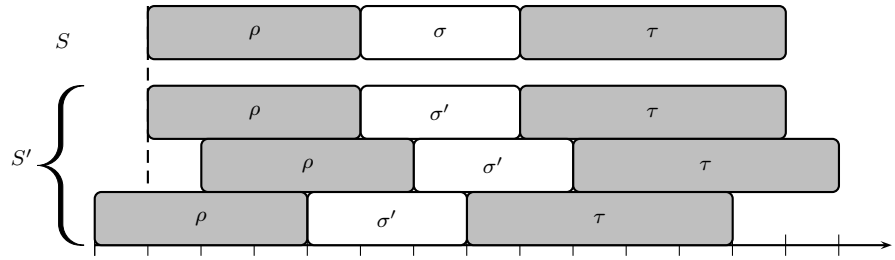

Fig. 8 If $\Phi_{\rho\circ\sigma'\circ\tau}(\alpha_{NI}(\rho\circ\sigma'\circ\tau)) < \Phi_{\rho\circ\sigma\circ\tau}(\alpha_{NI}(\rho\circ\sigma\circ\tau))$, then S is not optimal.

σ' of σ such that the sequence $\rho \circ \sigma' \circ \tau$ leads to a non-idling schedule S' with a lower cost than those of S, *i.e.* $\Phi_{\rho\circ\sigma'\circ\tau}(\alpha_{NI}(\rho\circ\sigma'\circ\tau)) < \Phi_{\rho\circ\sigma\circ\tau}(\alpha_{NI}(\rho\circ\sigma\circ\tau))$, then S is obviously not optimal. Note that depending on the jobs which start at their release dates in S, we can have $\alpha_{NI}(\rho\circ\sigma'\circ\tau) = \alpha_{NI}(\rho\circ\sigma\circ\tau)$, $\alpha_{NI}(\rho\circ\sigma'\circ\tau) > \alpha_{NI}(\rho\circ\sigma\circ\tau)$ or $\alpha_{NI}(\rho\circ\sigma'\circ\tau) < \alpha_{NI}(\rho\circ\sigma\circ\tau)$.

Consider any non-idling schedule S and consider a subset $X \subset N$ of jobs. Let us assume that we delay the whole schedule by δ time units, retaining exactly the same sequence of jobs. Note that this schedule is non-idling. Let us define $\overline{\Delta}_X(\delta)$ as any nondecreasing upper bound of the difference between the total cost of jobs belonging to X before and after delaying the schedule by δ time units, for any schedule S. Now, suppose that the whole schedule may be brought forward from δ time units without violating the non-idling constraint and retaining exactly the same sequence of jobs. Let us define $\underline{\Delta}_X(\delta)$ as any nondecreasing lower bound of the difference between the total cost belonging to X before and after bringing forward these jobs from δ time units, for any non-idling schedule S.

Given a sequence of jobs σ, we denote as $\bar{\sigma}$ the set of jobs not belonging to sequence σ, *i.e.* $\bar{\sigma} = \{l \in N | l \notin \sigma\}$. We can now describe sufficient conditions ensuring that a partial solution S cannot lead to an optimal solution:

Proposition 8 *Suppose that Φ is additively separable into nondecreasing functions of individual end times. Consider a partial schedule S and three complementary subsequences ρ, σ, τ in $F(S)$ or in $L(S)$, i.e. $\rho \circ \sigma \circ \tau = F(S)$ or $\rho \circ \sigma \circ \tau = L(S)$. Consider a sequence σ' such that σ' is a permutation of σ. If one of the following two conditions holds:*

1. $[\alpha_{NI}(\sigma') \leq est_{\sigma(1)}(S)] \wedge [\forall t / est_{\sigma(1)}(S) \leq t \leq lst_{\sigma(1)}(S), \Phi_{\sigma'}(t) < \Phi_\sigma(t)]$;
2. $[\alpha_{NI}(\sigma') > est_{\sigma(1)}(S)]$
   $\wedge [\Phi_{\sigma'}(\alpha_{NI}(\sigma')) < \Phi_\sigma(\alpha_{NI}(\sigma')) - \overline{\Delta}_{\bar{\sigma}}(\alpha_{NI}(\sigma') - est_{\sigma(1)}(S))]$
   $\wedge [\forall t / \alpha_{NI}(\sigma') \leq t \leq lst_{\sigma(1)}(S), \Phi_{\sigma'}(t) < \Phi_\sigma(t)]$;

then S cannot lead to an optimal solution.
Moreover, if the following condition holds:

3. $[\delta = \min(est_{\sigma(1)}(S) - \alpha_{NI}(\sigma'), \min_{i\in\bar{\sigma}}(est_i(S) - r_i)) > 0]$
   $\wedge [\Phi_{\sigma'}(est_{\sigma(1)}(S) - \delta) < \Phi_\sigma(est_{\sigma(1)}(S)) + \underline{\Delta}_{\bar{\sigma}}(\delta)]$;

then S cannot lead to an optimal semi-active solution.

Proof *Consider a partial solution S and let S' be any complete solution derived from solution S. Consider three complementary subsequences ρ, σ, τ of $F(S)$ or $L(s)$, i.e. $\rho \circ \sigma \circ \tau = F(S)$ or $\rho \circ \sigma \circ \tau = L(S)$ (see*

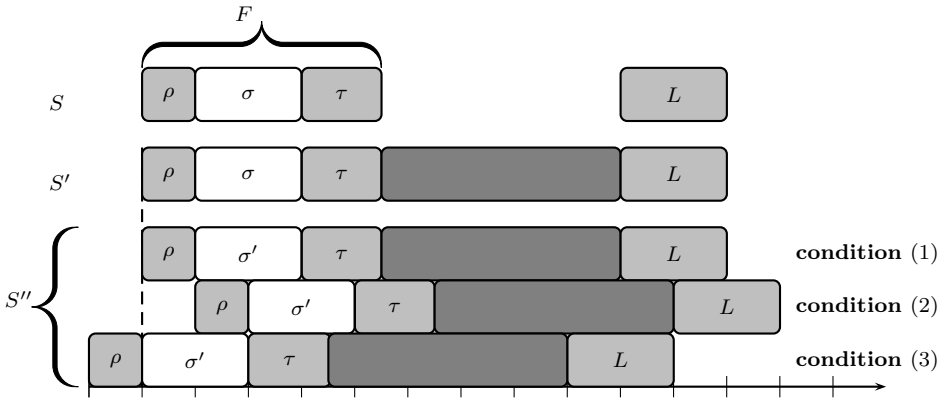

Fig. 9 S can not lead to an optimal solution.

Figure 9). Given that in S all jobs in σ are already sequenced, we must have $\alpha_{NI}(\sigma) \leq est_{\sigma(1)}(S)$ by definition. In S', the first job of the subsequence σ starts at time $start_{\sigma(1)}(S')$ with $lst_{\sigma(1)}(S) \geq start_{\sigma(1)}(S') \geq est_{\sigma(1)}(S) \geq \alpha_{NI}(\sigma)$.

Suppose that there exists a sequence σ' satisfying ***condition (1)****. Since $\alpha_{NI}(\sigma') \leq est_{\sigma(1)}(S) \leq start_{\sigma(1)}(S')$, rescheduling the jobs belonging to σ in the order of σ' from time $start_{\sigma(1)}(S')$ to obtain schedule S'' does not delay the other jobs and does not violate the non-idling constraint. Moreover, since $\forall t / est_{\sigma(1)}(S) \leq t \leq lst_{\sigma(1)}(S), \Phi_{\sigma'}(t) < \Phi_{\sigma}(t)$ holds, it follows that $\Phi_{\sigma'}(start_{\sigma(1)}(S')) < \Phi_{\sigma}(start_{\sigma(1)}(S'))$. Consequently, we obtain a schedule whose cost is strictly lower than $\Phi(S')$. Thus, S' is not optimal.*

Suppose that there exists a sequence σ' satisfying ***condition (2)****. Suppose that $start_{\sigma(1)}(S') \geq \alpha_{NI}(\sigma')$. Just as we did for the previous condition, we can prove that rescheduling the jobs belonging to σ in the order of σ' leads to a non-idling schedule better than S', which is consequently not optimal. Now, if $start_{\sigma(1)}(S') < \alpha_{NI}(\sigma')$, then rescheduling the jobs belonging to σ in the order of σ' leads to a schedule S'' in which all other jobs belonging to $\bar{\sigma}$ also are to be delayed by $\alpha_{NI}(\sigma') - start_{\sigma(1)}(S')$ time units. Since $\overline{\Delta}_{\bar{\sigma}}$ is nondecreasing, the additional cost of delaying these jobs is at most $\overline{\Delta}_{\bar{\sigma}}(\alpha_{NI}(\sigma') - start_{\sigma(1)}(S')) \leq \overline{\Delta}_{\bar{\sigma}}(est_{\alpha_{NI}(\sigma')} - est_{\sigma(1)}(S))$. Since it holds that $\Phi_{\sigma'}(\alpha_{NI}(\sigma')) + \overline{\Delta}_{\bar{\sigma}}(start_{\sigma(1)}(S') - \alpha_{NI}(\sigma')) \leq \Phi_{\sigma'}(\alpha_{NI}(\sigma')) + \overline{\Delta}_{\bar{\sigma}}(est_{\sigma(1)}(S) - \alpha_{NI}(\sigma')) < \Phi_{\sigma}(\alpha_{NI}(\sigma'))$, then we obtain a schedule whose cost is strictly lower than $\Phi(S')$. Thus, S' is not optimal.*

Suppose that there exists a sequence σ' satisfying ***condition (3)****. If S can lead to a semi-active schedule S', then it must hold that $start_{\sigma(1)}(S') = est_{\sigma(1)}(S') = \alpha_{NI}(\sigma)$ because $\min_{\{i \in \bar{\sigma}\}}(est_i(S) - r_i) > 0$, and at least one job in σ has to start at its release date (see Section 4). If we reschedule the jobs belonging to σ in the order of σ', we then obtain a non-idling schedule S'' which can be brought forward by $\delta \leq \min(est_{\sigma(1)}(S) - \alpha_{NI}(\sigma'), \min_{\{i \in \bar{\sigma}\}}(start_i(S') - r_i))$ time units. The benefit to be gained by bringing these jobs forward is therefore at least $\underline{\Delta}_{\bar{\sigma}}(\delta)$. We have $start_{\sigma(1)}(S') = est_{\sigma(1)}(S) - \delta$. Since $\Phi_{\sigma'}(est_{\sigma(1)}(S) - \delta) - \underline{\Delta}_{\bar{\sigma}}(\delta) < \Phi_{\sigma}(est_{\sigma(1)}(S))$, we obtain a schedule whose cost is strictly lower than $\Phi(S')$. Thus, S' is not optimal.*

While under Condition (3) the partial solution cannot lead to an optimal semi-active solution, it can lead to an optimal solution. Note that for regular criteria such as the total tardiness, an optimal schedule is not necessarily semi-active. This nuance is not needed for Conditions (1) and (2). These remarks explain why these cases are handled separately.

The previous proposition can be used to filter the sets $PF(S)$ and $PL(S)$ of a partial solution S. Suppose that ranking a job i belonging to $PF(S)$ (respectively, to $PL(S)$) immediately after $F(S)$ (respectively, immediately before $L(S)$) leads to a partial solution such that there exists a permutation σ' of a subsequence σ in $F(S) \circ (i)$ (respectively, $(i) \circ L(S)$) satisfying one of the above conditions, then i can be removed from $PF(S)$ (respectively, from $PL(S)$). Of course, enumerating all such possible sequences σ' is not relevant from a practical point of view. During the search with our enumerative method, we only consider sequences σ' recorded from previously encountered solutions. Using suited data structures, relevant recorded sequences can be found in an effective way. An example on how to consider such sequences is provided in the next section for the $1|r_i|\sum w_i C_i$ problem. Such sequences σ' could be also searched heuristically for example with a local search algorithm with a limited time complexity (see for example (18)). Values $\overline{\Delta}_X(\delta)$ and $\underline{\Delta}_X(\delta)$ should be established according to the criterion to be minimized.

7 Minimizing the total (weighted) completion time

In its classical version without the non-idling constraint, the one-machine total (weighted) completion time problem has been intensively addressed over the last two decades. The problems are denoted as $1|r_i|\sum C_i$ and $1|r_i|\sum w_iC_i$. In the case of identical release dates, both the unweighted and the weighted problems can easily be solved polynomially in $O(n \log n)$ time by applying the *WSPT* (Weighted Shortest Processing Time) priority rule, also known as Smith's rule (27): this consists of sequencing the jobs in nondecreasing order of their ratio of processing time to weight. The unweighted problem with release dates is NP-hard in the strong sense (22). Several authors have proved dominance properties and proposed a number of algorithms (9, 15, 16). Chu (11) proved several dominance properties and provided a branch-and-bound algorithm. Chand, Traub and Uzsoy used a decomposition approach to improve branch-and-bound algorithms (8). In its preemptive version, $1|r_i, pmtn|\sum C_i$ can be solved polynomially with the Shortest Remaining Processing Time (*SRPT*) rule (26): pieces of jobs are iteratively scheduled by choosing at each time a job with the shortest remaining processing time among the available and uncompleted jobs; this job is partially or totally scheduled until the time it is completed or another job becomes available. The solution of this problem is often used within branch-and-bound methods as a lower bound for the non-preemptive problem. However, Della Croce and T'Kindt (14) proposed an improved lower bound relying on the preemptive relaxation. The weighted case with release dates $1|r_i|\sum w_iC_i$ is NP-hard in the strong sense (25), even when preemption is allowed (21). Several dominance rules and branch-and-bound algorithms have been proposed (3, 4, 17, 24). Finally, Jouglet *et al.* (19) studied both the weighted and the unweighted cases (along with the more general problem $1|r_i|\sum (w_i)T_i$). They generalized and improved the above dominance rules which were used in a branch-and-bound method and in heuristic algorithms (20).

In this section we show how to apply the method and the techniques described in the previous sections to the $1, NI|r_i|\sum w_iC_i$ problem. In Section 7.1 we show that all optimal schedules of the problems are semi-active and active. In Section 7.2, we propose an application of the non-optimality sufficient condition of Proposition 8 and we describe a "no-good recording" technique for applying it. Some dominance rules specific to the studied problem are described in Section 7.3. Finally, techniques based on the lower bounds of the problems are described in Section 7.4.

7.1 Semi-active and active non-idling schedules

We saw in Section 4 and Section 5 that non-idling semi-active and active schedules are dominant for regular criteria. Note that for $1, NI|r_i|\sum w_iC_i$, any non-idling schedule which is not semi-active or active allows the possibility of a strict improvement of the value of the objective function. Thus, according to the following propositions, the properties are much stronger for the $1, NI|r_i|\sum w_iC_i$ problem:

Proposition 9 *All optimal non-idling schedules of the* $1, NI|r_i|\sum w_iC_i$ *problem are active.*

7.2 Non-optimal schedules

In this section we show how to express the non-optimality sufficient condition of Proposition 8 for the $1, NI|r_i|\sum w_iC_i$ problem, and we describe a "no-good recording" technique for applying it.

Because of the non-idling constraint (and in contrast to the classical problem), the following lemma holds and allows us to derive interesting properties for applying the non-optimality sufficient condition:

Lemma 1 *For any sequence* σ *and for any times* t *and* t' *such that* $min(t, t') \geq \alpha_{NI}(\sigma)$*, we have* $\Phi_\sigma(t) - \Phi_\sigma(t') = (t - t')\sum_{i\in\sigma} w_i$.

Proof Since $min(t, t') \geq \alpha_{NI}(\sigma)$*, the schedules obtained by scheduling jobs in the order of sequence* σ*, starting at time* t *and* t' *respectively, are non-idling. Without loss of generality, suppose that* $t' < t$*. Delaying a job* i *by* $t - t'$ *units of time increases its cost by* $(t - t')w_i$*, and bringing forward a job by* $t - t'$ *units of time decreases its cost by* $(t - t')w_i$.

The first and most obvious consequence is that $\overline{\Delta}_X(\delta) = \underline{\Delta}_X(\delta) = \delta\sum_{i\in X} w_i$ are valid and the tightest evaluations which can be taken according to their definition (see Section 6). Moreover, the following corollary is established:

Corollary 2 *Let* σ *and* σ' *be two sequences of the same set of jobs* X*. If for a given time such that* $t \geq \max(\alpha_{NI}(\sigma), \alpha_{NI}(\sigma'))$ *we have* $\Phi_\sigma(t) < \Phi_{\sigma'}(t)$*, then* $\forall t' \geq t$ *we have* $\Phi_\sigma(t') \leq \Phi_{\sigma'}(t')$.

Proof We have $\Phi_\sigma(t') = \Phi_\sigma(t) + (t'-t)\sum_{i\in X} w_i$ and $\Phi_{\sigma'}(t') = \Phi_{\sigma'}(t) + (t'-t)\sum_{i\in X} w_i$. Thus, $\Phi_\sigma(t) < \Phi_{\sigma'}(t) \Rightarrow \Phi_\sigma(t') < \Phi_{\sigma'}(t')$.

It will be noted that the following corollary of Lemma 1 allows us to compute $\Phi_\sigma(t)$ for all $t \geq \alpha_{NI}(\sigma)$, given $\Phi_\sigma(\alpha_{NI}(\sigma))$ and $\sum_{i\in\sigma} w_i$:

Corollary 3 *For any $t \geq \alpha_{NI}(\sigma)$, we have $\Phi_\sigma(t) = \Phi_\sigma(\alpha_{NI}(\sigma)) + (t - \alpha_{NI}(\sigma))\sum_{i\in\sigma} w_i$.*

As in Section 6, given a sequence of jobs σ, we denote as $\bar{\sigma}$ the set of jobs not belonging to sequence σ, *i.e.* $\bar{\sigma} = \{l \in N | l \notin \sigma\}$. Since we know that all optimal schedules are semi-active for $\Phi = \sum w_i C_i$ (which is additively separable into nondecreasing functions of individual end times), we see that the following proposition (which is a simplification of Proposition 8) also holds:

Proposition 10 *Consider a partial schedule S and three complementary subsequences ρ, σ, τ of $F(S)$ or $L(s)$, i.e. $\rho\circ\sigma\circ\tau = F(S)$ or $\rho\circ\sigma\circ\tau = L(S)$. If there exists a sequence σ' such that σ' is a permutation of σ and such that one of the 3 following conditions holds:*

1. $[\alpha_{NI}(\sigma') \leq est_{\sigma(1)}(S)] \wedge [\Phi_{\sigma'}(est_{\sigma(1)}(S)) < \Phi_\sigma(est_{\sigma(1)}(S))]$;
2. $[\alpha_{NI}(\sigma') > est_{\sigma(1)}(S)] \wedge [\Phi_{\sigma'}(\alpha_{NI}(\sigma')) + (\alpha_{NI}(\sigma') - est_{\sigma(1)}(S))\sum_{i\in\bar{\sigma}} w_i < \Phi_\sigma(est_{\sigma(1)}(S))]$;
3. $[\delta = \min(est_{\sigma(1)}(S) - \alpha_{NI}(\sigma'), \min_{i\in\bar{\sigma}}(est_i(S) - r_i)) > 0]$
   $\wedge[\Phi_{\sigma'}(est_{\sigma(1)}(S) - \delta) - \delta\sum_{i\in\bar{\sigma}} w_i < \Phi_\sigma(est_{\sigma(1)}(S))]$;

then S cannot yield an optimal solution.

As in (19), a no-good-recording technique (29) can be used, which "records", information about partial solutions encountered within the search tree. The characteristics of any partial solution S associated with a particular node of the search tree are stored in a "no-good" set. For a subsequence σ of $F(S)$ or $L(S)$, only the following data are saved: the set of jobs belonging to sequence σ, its associated earliest starting time $\alpha_{NI}(\sigma)$, the cost $\Phi_\sigma(\alpha_{NI}(\sigma))$ of this sequence when scheduled at $\alpha_{NI}(\sigma)$, and the total weight $\sum_{i\in\sigma} w_i$ of jobs belonging to σ. We store only the subsequences of $F(S)$ that include the last job of $F(S)$ and the subsequences of $L(S)$ that include the first job of $L(S)$, since the other sequences have already been stored at a previous node during the construction of S.

We use this information to filter the set of possible first jobs $PF(S)$ and $PL(S)$ of a partial solution S. For all $i \in PF(S)$ and for all sequences σ such that $F(S)\circ(i) = \rho\circ\sigma$, we seek a permutation σ' in the "no-good" set whose state meets the sufficient condition of Proposition 10. If such a state is found then scheduling a job i immediately after $F(S)$ yields a partial solution which is not optimal, and we can remove i from set $PF(S)$. Set $PL(S)$ is filtered in the same way.

A hash table combined with height-balanced binary search trees makes for an efficient use of the "no-good" set. It generally allows us to obtain a pertinent state of the set in $O(n\log n)$. A state of the set consists of a list of jobs sorted in lexicographic order coded in a vector, and a list of couples $(\Phi, \alpha_{NI}))$ corresponding to the different states of the nodes which have been visited within this set of jobs. Note that only non-comparable couples according to Proposition 10 can be kept. Suppose that we search a state with the same set of jobs belonging to sequence σ. 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 remains to look if a couple $(\Phi, \alpha_{NI}))$ of the state satisfies the sufficient condition of Proposition 10. After experimental tests, we noted that the amount of such (non-comparable) couples is very low. As shown with experimental results of Section 8, the search of relevant sequences σ' in the "no-good" set with such algorithm is very effective.

7.3 Dominance rules

In this section we describe dominance rules specific to $1, NI|r_i|\sum w_i C_i$. These dominance rules can be used to adjust time windows of jobs or to filter sets PF and PL. They all rely on the fact that when we have equal release dates an optimal sequence obeys the *WSPT* rule (27).

Proposition 11 *Given a partial solution S, if there exists a job i such that $\forall j \in NR\backslash\{i\}$, $\frac{p_i}{w_i} < \frac{p_j}{w_j}$ and $est_i(S) = \min_{j\in NR(S)} est_j(S)$, then in all optimal schedules that can be obtained from S, job i starts immediately after $F(S)$.*

Proof Let S' be an optimal schedule derived from S in which a job $j \neq i$ is scheduled just after $F(S)$ in S'. Let x be the job scheduled just before i in S' (eventually j). We have $est_i(S) \leq est_j(S) \leq start_j(S') \leq start_x(S') < start_i(S')$. Thus, jobs i and x can be interchanged without violating the non-idling constraint and without changing the starting times of all other jobs. Let S'' be the obtained schedule. We have $\Phi(S') - \Phi(S'') = w_x(start_x(S') + p_x) + w_i(start_x(S') + p_x + p_i) - w_i(start_x(S') + p_i) - w_x(start_x(S') + p_i + p_x) = w_i p_x - w_x p_i > 0$ since $p_i/w_i < p_x/w_x$. Consequently, the interchange leads to a schedule S'' with a strictly lower cost than $\Phi(S')$, which contradicts the fact that S' is optimal.

Proposition 12 *Given two jobs i and j such that $p_i = p_j$, $r_i \leq r_j$ and $(w_i > w_j) \vee (w_i = w_j \wedge i < j)$, then there exists an optimal schedule in which i is scheduled before j.*

Proof Let S be an optimal schedule in which j is scheduled before i. The two jobs can be interchanged without modifying the starting times of the other jobs since $p_i = p_j$ and $r_i \leq r_j$. We have $w_i \geq w_j$. Consequently, the interchange leads to a schedule with a cost at least as good as $\Phi(S)$ since $w_j end_j(S) + w_i end_i(S) \geq w_j end_i(S) + w_i end_j(S)$.

Proposition 13 *For all consecutive jobs i and j in an optimal schedule S, we have $p_i/w_i \leq p_j/w_j$ or $start_i(S) < r_j$.*

Proof Consider an optimal schedule S in which there exist two consecutive jobs i and j (i just before j) such that $p_i/w_i > p_j/w_j$ and $start_i(S) \geq r_j$. Jobs i and j can be interchanged without violating the non-idling constraint and without changing the start times of all other jobs. The difference of cost between S and the obtained schedule is equal to $w_i(start_i(S) + p_i) + w_j(start_i(S) + p_i + p_j) - w_j(start_i(S) + p_j) - w_i(start_i(S) + p_i + p_j) = w_j p_i - w_i p_j > 0$ since $p_i/w_i > p_j/w_j$. Consequently, the interchange leads to a schedule with a strictly lower cost than $\Phi(S)$, which contradicts the fact that S is optimal.

Proposition 14 *Given a partial solution S, if there exists a job $i \in NR(S)$ with $\delta = est_i(S) - eet_F(S)$ such that:*

$$\exists x \in F(S) / \begin{cases} (p_x > p_i \wedge w_x \leq w_i) \vee (p_x \geq p_i \wedge w_x < w_i) \\ r_i \leq est_x(S) + \delta \\ \min_{\{j \in F(S)/est_j(S) > est_x(S)\}} (est_j(S) + \delta - r_j) \geq p_x - p_i \end{cases}$$

then ranking job i immediately after $F(S)$ cannot lead to an optimal solution.

Proof Suppose that there exists an optimal schedule S' derived from S in which i is scheduled just after $F(S)$. Thus, $F(S)$ ends not earlier than $est_i(S)$ in S', hence $\forall j \in F(S), start_j(S') \geq est_j(S) + \delta$ holds and $\min_{\{j \in F(S)/est_j(S) > est_x(S)\}} (start_j(S') - r_j) \geq \min_{\{j \in F(S)/est_j(S) > est_x(S)\}} \{est_j(S) + \delta - r_j\} \geq p_x - p_i$. Consequently, jobs x and i can be interchanged without violating the non-idling constraint since all jobs between x and i can be scheduled $p_x - p_i$ units of time earlier. Note that in the obtained schedule, i ends at $end_x(S') - p_x + p_i$ and x ends at $end_i(S')$. Note that the start times of the other jobs (before x and after i) do not change. Therefore, the difference between the cost of S' and the cost of the obtained schedule is greater than or equal to $w_x end_x(S') + w_i end_i(S') - w_i(end_x(S') - p_x + p_i) - w_x end_i(S') = (end_i(S') - end_x(S')) \cdot (w_i - w_x) + w_i(p_x - p_i) > 0$ since $(p_x > p_i \wedge w_x \leq w_i) \vee (p_x \geq p_i \wedge w_x < w_i)$ holds. As a result of the interchange the total cost strictly decreases. This contradicts the fact that S' is optimal.

Proposition 15 *Given a partial solution S, let x and i be two jobs in the sequence $F(S)$ such that $(p_x > p_i \wedge w_x \leq w_i) \vee (p_x \geq p_i \wedge w_x < w_i)$ and $est_x(S) < est_i(S)$ (i.e. x is before i in sequence $F(S)$). Let δ be equal to:*

$$\delta = \max \begin{cases} r_i - est_x(S) \\ p_x - p_i - \min_{\{j \in F(S)/est_x(S) < est_j(S) < est_i(S)\}} (est_j(S) - r_j) \end{cases}$$

then in any optimal schedule S' which may be derived from S, we have $\forall j \in F(S), start_j(S') < est_j(S) + \delta$.

Proof Suppose that there exists an optimal schedule S' derived from S in which $\exists k \in F(S)$ with $start_k(S') \geq est_k(S) + \delta$. It holds $\forall j \in F(S), start_j(S') \geq est_j(S) + \delta$. Thus, we have $r_i \leq est_x(S) + \delta \leq start_x(S')$. Moreover, we have $\min_{\{j \in F(S)/est_x(S) < est_j(S) < est_i(S)\}} (start_j(S') - r_j)$ which is greater than or equal to $\min_{\{j \in F(S)/est_x(S) < est_j(S) < est_i(S)\}} (est_j(S) + \delta - r_j) \geq p_x - p_i$. Consequently, jobs x and i can be interchanged without violating the non-idling constraint since all jobs between x and i can be scheduled $p_x - p_i$ units of time earlier. Note that the starting times of the other jobs (before x and after i) do not change. Therefore, the difference between the cost of S' and the cost of the obtained schedule is greater than or equal to $w_x end_x(S') + w_i end_i(S') - w_i(end_x(S') - p_x + p_i) - w_x end_i(S') = (end_i(S') - end_x(S')) \cdot (w_i - w_x) + w_i(p_x - p_i) > 0$ since $(p_x > p_i \wedge w_x \leq w_i) \vee (p_x \geq p_i \wedge w_x < w_i)$ holds. As a result of the interchange the total cost strictly decreases. This contradicts the fact that S' is optimal.

Proposition 16 *Given a partial solution S, if there exists a job $i \in NR(S)$ with $\delta = \min_{j \in NR(S)\setminus\{i\}} est_j(S) - eet_F(S)$ such that:*

$$\exists x \in L(S) / \begin{cases} (p_x < p_i \wedge w_x \geq w_i) \vee (p_x \leq p_i \wedge w_x > w_i) \\ r_x \leq est_L(S) + \delta - p_i \\ \min_{\{j \in L(S) / est_j(S) < est_x(S)\}} (est_j(S) + \delta - r_j) \geq p_i - p_x \end{cases}$$

then ranking job i immediately before $L(S)$ cannot lead to an optimal solution.

Proof Symmetrically analogous to the proof of Proposition 14.

Proposition 17 *Given a partial solution S, let i and x be two jobs of sequence $L(S)$ such that $(p_x < p_i \wedge w_x \geq w_i) \vee (p_x \leq p_i \wedge w_x > w_i)$ and $est_i(S) < est_x(S)$ (i.e. i is before x in sequence $L(S)$). Let δ be equal to:*

$$\delta = \max \begin{cases} r_x - est_i(S) \\ p_i - p_x - \min_{\{j \in L(S) / est_i(S) < est_j(S) < est_x(S)\}} (est_j(S) - r_j) \end{cases}$$

then in any optimal schedule S' which may be derived from S we have $\forall j \in L(S), start_j(S') < est_j(S) + \delta$.

Proof Symmetrically analogous to the proof of Proposition 15.

7.4 Lower bounds: cuttings and adjustments

In this section we describe the lower bounds which are used in our branch-and-bound algorithm. In Section 7.4.1 we show how the $WSPT$ rule can be efficiently used to filter sets $PF(S)$ and $PL(S)$ in $O(n)$ time if the $WSPT$ order is known. In Section 7.4.2 we recall other lower bounds which are also used in the branch-and-bound method.

7.4.1 Relaxing the release dates

Given a partial schedule S, the $WSPT$ rule can be easily used to compute a lower bound of the total cost of S by relaxing the earliest starting times of jobs belonging to $NR(S)$. Let $WSPT_X(t)$ be the value of the cost function obtained by applying the $WSPT$ rule to jobs in set X by relaxing their release dates to t. Since we have $est_L(S) = eet_F(L) + \sum_{i \in NR(S)} \{p_i\}$, then the value $LB^{WSPT}(S) = \Phi_{F(S)}(est_F(S)) + WSPT_{NR(S)}(eet_F(S)) + \Phi_{L(S)}(est_L(S))$ is a lower bound of the cost of the partial solution S. If this bound is strictly greater than $\bar{\Phi}$, then the partial solution S cannot lead to a better solution than one which has already been found. Note that if the $WSPT$ sequence of jobs belonging to $NR(S)$ (taking the original earliest starting times) leads to a non-idling schedule starting at time $eet_F(S)$, then $LB^{WSPT}(S)$ is an upper bound of the cost of the partial solution S, and jobs belonging to $NR(S)$ can be automatically scheduled according to the $WSPT$ order. Note also that delaying a complete schedule by one time unit increases the cost of the objective function of $\sum_{i \in N} w_i$. A complete schedule derived from S will have a total cost of at least $LB^{WSPT}(S)$ and cannot end earlier than $Eet(S)$. Therefore, the latest completion time $Let(S)$ of the partial solution can be adjusted to $\min(Let(S), Eet(S) + (\bar{\Phi} - LB^{WSPT}(S))/\sum_{i \in N} w_i)$.

From now on, to simplify the presentation, we shall assume that jobs in $NR(S)$ are indexed from 1 to $|NR(S)|$ according to the $WSPT$ rule. Let $\Phi_i^{WSPT}(S) = w_i(eet_F(S) + \sum_{j=1}^{i} p_j)$ be the contribution of job i to the lower bound $WSPT_{NR(S)}(eet_F(S))$.

Suppose now that a job i belonging to $NR(S)$ is scheduled in the first position immediately after $F(S)$. If $est_i(S) > eet_F(S)$ then the end time of the last job of sequence $F(S)$ will be adjusted to be greater than or equal to $est_i(S)$ to satisfy the non-idling constraint, right-shifting the whole sequence by at least $\delta_i = eet_F(S) - est_i(S) > 0$ time units. Thus, in the obtained schedule, sequence $F(S)$ will not end before $eet_F(S) + \delta_i$. A lower bound of the cost of a solution which can be obtained from S if i is scheduled immediately after $F(S)$ then becomes $LB_{i \to first}^{WSPT}(S) = \Phi_{F(S)}(est_F(S) + \delta_i) + w_i(eet_F(S) + \delta_i + p_i) + WSPT_{NR(S)\setminus\{i\}}(eet_F(S) + \delta_i + p_i) + \Phi_{L(S)}(est_L(S) + \delta_i)$. Fortunately, this new lower bound can easily be computed from $LB^{WSPT}(S)$. The following relation holds: $LB_{i \to first}^{WSPT}(S) = LB^{WSPT}(S) - \Phi_i^{WSPT}(S) + (eet_F(S) + \delta_i + p_i)w_i + (\delta_i + p_i)\sum_{j=1}^{i-1}(w_j) + \delta_i \sum_{j=i+1}^{|NR(S)|}(w_j) + \delta_i \sum_{j \in F(S) \cup L(S)}(w_j)$. It corresponds to the initial bound with corrections according to the insertion of job i before the other jobs (see Lemma 1). Now, if $LB_{i \to first}^{WSPT}(S)$ is strictly greater than $\bar{\Phi}$, then i cannot be scheduled immediately after $F(S)$, and it can be removed from $PF(S)$.

Algorithm 4: Evaluating, filtering and adjusting a partial solution with the *WSPT* rule.

Data: A partial solution S (*cf.* Figure 1), jobs being assumed to be sorted in nondecreasing order of the processing time to weight ratio

$LB^{WSPT}(S) \leftarrow \Phi_{F(S)}(est_F(S)) + \Phi_{L(S)}(est_L(S))$;
$t \leftarrow eet_F(S)$;
for $i \leftarrow 1$ *to* n **do**
  if $i \in NR(S)$ **then**
    $t \leftarrow t + p_i$;
    $\Phi_i^{WSPT}(S) \leftarrow t \cdot w_i$;
    $LB^{WSPT}(S) \leftarrow LB^{WSPT}(S) + \Phi_i^{WSPT}(S)$;
if $LB^{WSPT}(S) > \bar{\Phi}$ **then** a backtrack is triggered;
$W_N \leftarrow \sum_{j \in N} \{w_j\}$;
$W_R \leftarrow \sum_{j \in F(S) \cup L(S)} w_j$;
$W_A \leftarrow 0$;
$W_B \leftarrow \sum_{j \in NR(S)} w_j$;
$earliest\_job \leftarrow$ the job belonging to $NR(S)$ with the smallest earliest start time breaking times by choosing the job with the smallest index ;
$earliest\_job' \leftarrow$ the job belonging to $NR(S) \backslash \{earliest\_job'\}$ with the smallest earliest start time breaking times by choosing the job with the smallest index ;
for $i \leftarrow 1$ *to* n **do**
  if $i \in NR(S)$ **then**
    $W_B \leftarrow W_B - w_i$;
    if $i \in PF(S)$ **then**
      $\delta_i \leftarrow est_i(S) - eet_F(S)$;
      $LB_{i \rightarrow first}^{WSPT}(S) \leftarrow LB^{WSPT}(S) - \Phi_i^{WSPT}(S) + (eet_F(S) + \delta_i + p_i)w_i + (\delta_i + p_i)W_A + \delta_i(W_B + W_R)$;
      if $LB_{i \rightarrow first}^{WSPT}(S) > \bar{\Phi}$ **then** $PF(S) \leftarrow PF(S) \backslash \{i\}$;
    if $i \in PL(S)$ **then**
      if $i \neq earliest\_job$ **then** $\delta_i \leftarrow est_{earliest\_job}(S) - eet_F(S)$;
      else $\delta_i \leftarrow est_{earliest\_job'}(S) - eet_F(S)$
      $LB_{i \rightarrow last}^{WSPT}(S) \leftarrow LB^{WSPT}(S) - \Phi_i^{WSPT}(S) + (est_L(S) + \delta_i)w_i + \delta_i(W_R + W_A) + (\delta_i - p_i)W_B$;
      if $LB_{i \rightarrow first}^{WSPT}(S) > \bar{\Phi}$ **then** $PL(S) \leftarrow PL(S) \backslash \{i\}$;
      else $let_i(S) \leftarrow \min(let_i(S), est_L(S) + \delta_i + (\bar{\Phi} - LB_{i \rightarrow last}^{WSPT}(S)) / \sum_{j \in N} w_j$;
    $W_A \leftarrow W_A + w_i$;

Similarly, suppose that a job i belonging to $NR(S)$ is scheduled in the last position immediately before $L(S)$. Sequence $F(S)$ cannot end earlier than $eet_F(S) + \delta_i$ and sequence $L(S)$ cannot start earlier than $est_L(S) + \delta_i$, in which $\delta_i = \min_{j \in NR(S) \backslash \{i\}} est_j(S) - eet_F(S)$, otherwise the non-idling constraint cannot be satisfied. A lower bound $LB_{i \rightarrow last}^{WSPT}(S)$ of the cost of a solution which can be obtained from S if i is scheduled immediately before $L(S)$ can easily be computed from $LB^{WSPT}(S)$, since the following relation holds: $LB_{i \rightarrow last}^{WSPT}(S) = LB^{WSPT}(S) - \Phi_i^{WSPT}(S) + \delta_i \sum_{j=1}^{i-1}(w_j) + (\delta_i - p_i) \sum_{j=i+1}^{|NR(S)|}(w_j) + \delta_i \sum_{j \in F(S) \cup L(S)}(w_j) + w_i(est_L(S) + \delta_i)$. Now, if $LB_{i \rightarrow last}^{WSPT}(S)$ is strictly greater than $\bar{\Phi}$, then i cannot be scheduled immediately before $L(S)$, and it can be removed from $PL(S)$.
Where $LB_{i \rightarrow last}^{WSPT}(S)$ is not strictly greater than $\bar{\Phi}$, we can nevertheless use this lower bound to adjust the latest completion time of job i. If i is scheduled immediately before $L(S)$ and completes at time $t \geq est_L(S) + \delta_i$, then a lower bound of the solution is $LB_{i \rightarrow last}^{WSPT}(S) + (t - est_L(S) - \delta_i) \sum_{j \in N} w_j$, since it follows that all jobs need to be delayed by $t - eet_F(S) - \delta_i$ time units (see Lemma 1). This bound has to be lower than or equal to $\bar{\Phi}$. Thus, job i cannot end after $est_L(S) + \delta_i + (\bar{\Phi} - LB_{i \rightarrow last}^{WSPT}(S)) / \sum_{j \in N} w_j$.

Considering that the *WSPT* order of jobs is known (for example, it is computed only once in $O(n \log n)$ at the beginning of the branch and bound algorithm), and in the light of the above remark, Algorithm 4 allows us to compute in $O(n)$ the lower bound $LB^{WSPT}(S)$, along with all lower bounds $LB_{i \rightarrow first}^{WSPT}(S)$ and $LB_{i \rightarrow last}^{WSPT}(S)$. These lower bounds are then used to trigger a backtrack or to adjust the time-windows of jobs.

7.4.2 Preemption and splitting

In addition, we use other lower bounds from the literature according to the criterion we want to minimize, in order to filter sets $PF(S)$ and $PL(S)$ for jobs which cannot be eliminated by the *WSPT* rule. For the $1|r_i| \sum w_i C_i$ problems, we use the lower bound of Belouadah, Posner and Potts (3), which is based on job splitting. For the $1|r_i| \sum C_i$ problem we use the SRPT rule (26).

8 Experimental results

All the techniques presented in this article have been incorporated into our branch-and-bound method implemented on top of the Ilog Solver and Ilog Scheduler. All experimental results were computed on a Dell Latitude D620 PC with a Genuine Intel T2600 2.16GHz CPU, running Windows Vista Professional.

The instances are generated using the same scheme as the test problems of Hariri and Potts (17) and, Belouadah, Posner and Potts (3) for the classical problems. For each job i, a processing time p_i is generated from the uniform distribution $[1, 100]$ and a weight w_i from the uniform distribution $[1, 10]$ for the weighted case. For a size $n = \{10, 20, \ldots, 100\}$ of problem, an integer release date r_i for each job J_i is generated from the uniform distribution $[0, 50.5 \cdot n \cdot R]$, where R controls the range of the distribution. For each selected value of n, five problems are 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 50 problems for each value of n.

In Table 1, the efficiency of the techniques described in the previous sections is studied on instances of size $n = 50$ jobs for the $1, NI|r_i|\sum C_i$ problem. For this experiment, we used a branching strategy in which the next job to be scheduled is chosen always from among the jobs in PF: the job in PF which yields the state with the smallest lower bound is ranked immediately after F. First, the results of the branch and bound method using all of the described techniques are shown ("all tech."). Then the different techniques are removed one by one to test their global contribution to the efficiency of the method. Algorithm 1, for propagating the non-idling constraint, is shown as "NI". Algorithm 2 and Algorithm 3, for propagating the non-idling semi-active and active constraints, are shown as "SAA". The no-good-recording technique, described in Section 7.2, is shown as "NGR". The application of the dominance rules of Section 7.3 are shown as "DR". Finally the different lower bounds are shown as WSPT and SRPT. For the different configurations of the branch and bound method and for each value of parameter R, the average number of generated nodes ("*nodes*") and the average computation time in seconds ("$cpu(s)$") over the 5 generated instances are shown. A time limit of 200 seconds was used. The number of instances over 5 which were not solved within the time limit is given in brackets. In this case, the time limit and the number of nodes generated within this time limit are used for the statistics. We can see that any of the techniques improves the behavior of the branch and bound method. Some techniques, such as SAA or WSPT, improve only the computation time. They adjust the state of the different variables more efficiently than other techniques which can perform the same deductions but at a higher computation time cost. Some others improve the behavior of the branch and bound method, in terms of both search space and computation time. The propagation of the non-idling constraint and the use of the SRPT lower bound would appear to be particularly effective. However, using only these two techniques is not really efficient, as shown by the column "NI + SRPT", which corresponds to the results obtained by the branch and bound procedure using only these two techniques.

In Table 2, different branching strategies are studied with instances of size $n = 50$ jobs for the $1, NI|r_i|\sum C_i$ problem, using all the described techniques. Column F shows the results obtained by the method in which the next job to be scheduled is chosen always from among the jobs in PF. Column L shows the results obtained by the method in which the next job to be scheduled is chosen always from among the jobs in PL. Column FL shows the results obtained by the method in which the next job to be scheduled is chosen from among the jobs in both PF and PL. In each configuration, the job which yields the state with the smallest lower bound is placed in the considered subset of jobs. For the different configurations of the branch and bound method and for each value of parameter R, the average number of generated nodes ("*nodes*") and the average computation time in seconds ("$cpu(s)$") over the 5 generated instances are given. It can be seen that the method F is significantly better on average than the methods L and FL. However, for instances with a large R ($R \geq 1.5$), we can see that the method L is more efficient. This can be explained by the fact that the parameter has a lot of influence on the starting time of the whole schedule. Recall that in an optimal (semi-active) solution, at least one job starts at its release date. For a given schedule, let μ be the first job starting at its release date in the schedule. It is this job which fixes the starting time of the entire schedule. With a small value of R, instances with μ located among the first jobs of the schedule often give the best solutions, because the range of release dates is small. Consequently, for these instances, it is often better to choose the next job to be scheduled from among those in PF, so as to make the starting time of the whole schedule the earliest possible. Conversely, when R is higher, μ is often located at the end of the schedule, and it may be more effective to chose the next job to be scheduled from among jobs belonging to PL. Note that the method FL does not represent an interesting compromise. In order to improve the performance of the branch and bound procedure, we compute two descriptors λ_1 and λ_2 for each instance. They allow us to decide the branching strategy which has a good chance of being the most efficient. Computing α_{NI} for a given instance, let χ_{NI} be the index of the last job necessitating a shift to the right of the whole schedule if the jobs are sorted in nondecreasing order

	all tech.		**all tech. - NI**		**all tech. - SAA**		**all tech. - NGR**	
R	*nodes*	*cpu(s)*	*nodes*	*cpu(s)*	*nodes*	*cpu(s)*	*nodes*	*cpu(s)*
0.2	23	0.18	23	0.11	23	0.16	27	0.23
0.4	141	0.93	141	0.41	141	0.90	216	1.6
0.6	1163	4.2	1237	2.5	1163	4.3	4097	20
0.8	914	2.8	22910	44(1)	914	2.9	8960	29
1	801	2.1	5423	7.5	801	2.2	9969	32
1.25	3456	6.7	43246	57(1)	3456	7.9	26237	52(1)
1.5	4524	11.7	30539	47(1)	4524	14.5	12794	37
1.75	3162	7.5	3450	5.0	3162	9.5	9320	32
2	1901	5.3	21438	35	1901	6.7	5431	18
3	842	1.9	1155	1.5	842	2.4	2026	6.9
av.	1693	4.3	12956	20(3)	1693	5.1	7908	23(1)
	all tech. - DR		**all tech. - SRPT**		**all tech. - WSPT**		**NI + SRPT**	
R	*nodes*	*cpu(s)*	*nodes*	*cpu(s)*	*nodes*	*cpu(s)*	*nodes*	*cpu(s)*
0.2	31	0.32	1046	4	23	0.19	250	1.0
0.4	255	1.5	55336	180	141	0.72	14174	57
0.6	1747	7.6	70159	167	1163	3.2	44541	138(3)
0.8	1147	5.2	90462	200(5)	915	2.4	46720	163(3)
1	1108	4.2	96033	200(5)	801	1.7	34684	200(5)
1.25	7317	29	77230	200(5)	3460	7.2	26541	167(4)
1.5	19597	72(1)	89008	200(5)	4526	15.0	38109	200(5)
1.75	9577	26	85747	200(5)	3164	10.2	49293	200(5)
2	14890	42	94137	200(5)	1906	8.0	35494	200(5)
3	32754	60(1)	115155	200(5)	846	2.5	64581	200(5)
av.	8842	25(2)	77431	176(42)	1694	5.1	35439	153(35)

Table 1 Testing the efficiency of the techniques. $1, NI|r_i|\sum C_i$, $n = 50$ jobs

	F		L		FL		F or L						$RTTO$	
R	*nodes*	*cpu(s)*	*nodes*	*cpu(s)*	*nodes*	*cpu(s)*	*nodes*	*cpu(s)*	nb_F	nb_L	λ_1	λ_2	*nodes*	*cpu(s)*
0.2	23	0.18	49035	571	49	0.33	23	0.18	5	0	0.00	0.00	43	0.24
0.4	141	0.93	9030	69	2788	12	141	0.93	5	0	0.00	0.00	238	1.01
0.6	1163	4.2	25833	187	42126	123	1163	4.2	5	0	0.02	0.02	1820	5.41
0.8	914	2.8	5934	41	7231	23	914	2.8	5	0	0.16	0.03	1830	4.74
1	801	2.1	1862	14	3431	8.7	835	2.5	4	1	0.21	0.05	1358	2.22
1.25	3456	6.7	1266	7.7	13474	34	1198	7.0	1	4	0.75	0.19	5069	7.38
1.5	4524	12	750	5.2	67063	169	750	5.2	0	5	0.92	0.45	5748	10.54
1.75	3162	7.5	316	1.8	39470	129	316	1.8	0	5	0.97	0.76	5399	9.58
2	1901	5.3	466	2.6	1930	6.8	466	2.6	0	5	0.96	0.94	2778	5.18
3	842	1.9	112	0.66	1299	3.5	112	0.66	0	5	0.97	2.0	927	1.46
av.	1693	4.3	9460	90	17886	51	592	2.8					2521	4.78

Table 2 Testing branching strategies

of release dates (see Algorithm 1). The value $\lambda_1 = \chi_{NI}/n$ is then the ratio between this value and n. A high value of λ_1 means that the starting time of a lot of jobs is delayed because of this job. The value $\lambda_2 = (\alpha_{NI} - \min_{i\in N}\{r_i\})/(\min_{i\in N}\{r_i\} + \sum_{i\in N}\{p_i\})$ represents the distance between the minimum release date and the α_{NI} value of the instance normalized by the sum of the processing times added to the minimum release date. Thus, the higher the values λ_1 and λ_2, the greater the number of solutions with a good chance of having μ among the last jobs of the schedule. Columns "F or L" show results obtained by the method using strategy F if $\lambda_1 \leq .5$ or $\lambda_2 \leq 0.1$, and strategy L otherwise. For each parameter R we also give the average values for λ_1 and λ_2, in order to justify this choice. Columns nb_F and nb_L show the number of times strategies F and L respectively were chosen. We can see that the results are significantly improved. However, note that this way of choosing F or L does not select the best strategy in several cases.

We have also tested the "Reduce-To-The-Opt" algorithm of Wolf (30). This method uses a strategy similar to the "F" strategy since it also builds a sequence of jobs by iteratively schedule jobs at the end of a partial sequence: considering that **the start time of the whole sequence is fixed**, this method relies on the fact that only jobs which are available can be scheduled at the end of the partial schedule. Thus, at each node of the search tree, an unscheduled job among ones with the smallest earliest start time is scheduled as soon as possible just after the partial sequence. Note that contrary to the method "F", the start time of the job which is scheduled is known at the time it is sequenced. Therefore, the advantage of this method is that propagation algorithms are more efficient and faster since the start time of scheduled jobs and the end of the whole schedule are known. The drawback is that the method has to be run on each possible start time of the sequence to find the optimum. As for our method, the "Reduce-To-The-Opt" algorithm is not able to solve most of the instances of 50 jobs within the time limit. However, except for the

	20				30				40			
	nodes		*cpu*(*s*)		*nodes*		*cpu*(*s*)		*nodes*		*cpu*(*s*)	
R	*av.*	*max*	*av.*	*max*	*av.*	*max*	*av.*	*max*	*av.*	*max*	*av.*	*max*
0.2	9.4	21	0.03	0.06	11	19	0.04	0.05	31	73	0.19	0.34
0.4	9.2	13	0.03	0.05	38	86	0.12	0.22	185	534	0.58	1.6
0.6	13	18	0.01	0.02	43	85	0.12	0.27	132	327	0.49	1.2
0.8	72	159	0.11	0.20	79	115	0.15	0.23	207	429	0.53	0.97
1	24	39	0.05	0.06	130	300	0.25	0.45	816	1485	3.7	8.1
1.25	17	25	0.03	0.06	48	60	0.15	0.20	933	3872	3.5	14
1.5	24	58	0.04	0.11	41	66	0.12	0.19	222	369	0.98	1.6
1.75	18	41	0.03	0.06	71	163	0.19	0.39	230	447	0.81	1.4
2	22	38	0.04	0.06	41	66	0.13	0.22	170	290	0.72	1.4
3	20	51	0.03	0.08	66	176	0.18	0.47	87	154	0.33	0.53
av.	23	159	0.04	0.20	57	300	0.1	0	301	3872	1.2	14

	50				60				70			
	nodes		*cpu*(*s*)		*nodes*		*cpu*(*s*)		*nodes*		*cpu*(*s*)	
R	*av.*	*max*	*av.*	*max*	*av.*	*max*	*av.*	*max*	*av.*	*max*	*av.*	*max*
0.2	23	37	0.18	0.30	60	133	0.6	1.2	67	105	1.0	1.5
0.4	141	318	0.93	1.8	180	377	1.4	2.6	1166	3363	10	25
0.6	1163	2623	4.2	11	855	1448	4.7	7.8	12610	39869	71	228
0.8	914	2976	2.8	9	2712	9868	11	38	5702	22581	28	108
1	835	2045	2.5	4.3	1160	3267	5.1	10	6091	12336	37	67
1.25	1198	2197	7.0	11	1427	3259	12	34	2669	5992	29	60
1.5	750	1883	5.2	15	1560	3398	13	29	8755	32643	108	444
1.75	316	625	1.8	3.5	804	2035	6.7	17	4990	14992	41	125
2	466	889	2.6	5.0	3882	9506	29	77	2204	7093	26	88
3	112	192	0.66	1.3	552	1656	4.2	13	1017	2299	8.7	18
av.	592	2976	2.8	15	1319	9868	8.8	77	4527	39869	36	444

	80				90				100			
	nodes		*cpu*(*s*)		*nodes*		*cpu*(*s*)		*nodes*		*cpu*(*s*)	
R	*av.*	*max*	*av.*	*max*	*av.*	*max*	*av.*	*max*	*av.*	*max*	*av.*	*max*
0.2	124	260	2.4	5.2	422	1590	11	40	197	352	5.6	12
0.4	1226	1998	14	23	1868	4988	25	61	4075	10368	62	167
0.6	1505	2736	14	22	7353	22961	68	208	16196	30234	204	421
0.8	5816	20364	34	108	23458	58804	158	326	36664	162080	286	1182
1	9077	28657	60	121	36270	97720	187	509	54253	230132	564	2285
1.25	83776	324210	1274	4852	85712	370661	1171	**(1)**	93424	167336	1576	3055
1.5	56815	153608	606	1374	62879	158854	1161	2977	74141	213885	1170	2666
1.75	8342	13659	118	199	48634	123891	754	2331	57378	190245	1267	4282
2	6495	12811	105	224	58540	131611	670	1384	16191	48012	312	828
3	9750	36162	87	260	5588	12132	109	228	4926	9920	113	177
av.	18293	324210	231	4852	33072	370661	431	**(1)**	35744	230132	556	4282

Table 3 $1, NI|r_i|\sum C_i$: main results

propagation of the non-idling constraint with Algorithm 1 which is useless since the start time of the whole sequence is known, all other techniques described in the article are also efficient with the "Reduce-To-The-Opt" method to reduce the search space. The results obtained by using the "Reduce-To-The-Opt" method and our techniques are reported on column "*RTTO*". As for our method, we have used an incremental search and the job yielding the state with the smallest lower bound is chosen to be scheduled at each node of the search tree. We can see that the number of nodes generated with the "Reduce-To-The-Opt" algorithm is higher than if we use our algorithm with the "F" strategy. However, from a practical point of view, computational times are not so different, showing that propagation stages run faster when the start time of the sequence is known. Whereas our algorithm using the "F" strategy seems to be slightly better on the average, the "Reduce-To-The-Opt" algorithm is competitive on a non-negligible part of the instances. Thus, better results could surely be obtained by combining the two algorithms. Note that we have also tried the "Reduce-To-The-Opt" algorithm by using "L", "FL" and "F or L" strategies. Similar observations and conclusions than for our method can be done. As suggested in (30), we have also tried a dichotomic bounding strategy concerning the value of the objective function. However, the results were decidedly lower.

In Table 3 and Table 4, general results of the branch and bound method for $n = \{10, 20, \ldots, 100\}$ are provided for the two studied criteria. The branching strategy "F or L" is used. For each instance size and for each value of R the average ("*av.*") and the maximum ("*max*") computation times in seconds ("*cpu*(*s*)") are shown, along with the average and the maximum number of generated nodes ("*nodes*") over the five generated instances. A time limit of 5000 seconds was used. Some instances are not solved within the time limit. In this case the number of unsolved instances is shown in brackets instead of the maximum

	10				20				30			
	nodes		*cpu(s)*		*nodes*		*cpu(s)*		*nodes*		*cpu(s)*	
R	*av.*	*max*	*av.*	*max*	*av.*	*max*	*av.*	*max*	*av.*	*max*	*av.*	*max*
0.2	2.0	3.0	0.02	0.02	7.4	13	0.04	0.08	11	22	0.08	0.14
0.4	3.6	6.0	0.02	0.03	18	33	0.05	0.11	33	76	0.16	0.30
0.6	8.8	21	0.02	0.03	28	73	0.07	0.17	91	215	0.37	0.77
0.8	7.4	17	0.01	0.03	42	82	0.09	0.13	179	616	0.54	1.8
1	6.6	12	0.01	0.02	50	97	0.13	0.22	583	1231	1.9	4.2
1.25	5.4	14	0.02	0.06	62	100	0.17	0.27	215	343	1.1	1.9
1.5	4.8	10	0.01	0.03	145	385	0.27	0.67	68	143	0.34	0.77
1.75	7.6	15	0.02	0.05	56	121	0.15	0.31	177	357	0.63	1.2
2	6.4	10	0.02	0.05	53	123	0.12	0.22	335	914	0.90	2.3
3	3.8	7.0	0.02	0.03	32	81	0.08	0.14	41	83	0.16	0.34
av.	5.6	21	0.02	0.06	49	385	0.12	1	173	1231	0.62	4

	40				50				60			
	nodes		*cpu(s)*		*nodes*		*cpu(s)*		*nodes*		*cpu(s)*	
R	*av.*	*max*	*av.*	*max*	*av.*	*max*	*av.*	*max*	*av.*	*max*	*av.*	*max*
0.2	37	75	0.33	0.50	104	255	1.2	2.8	358	1320	4.1	13
0.4	93	184	0.71	1.3	915	1369	8.0	13	568	1231	6.6	14
0.6	67	107	0.45	0.77	2222	5421	15	31	8765	24469	60	143
0.8	1675	3947	6.9	16	2026	3571	10	20	9265	28639	58	185
1	1408	3171	9.5	21	3424	7678	24	49	66153	289781	470	1849
1.25	1168	2519	7.5	16	8525	29475	67	239	72993	202095	686	1567
1.5	1510	3235	8.3	17	11107	44137	95	371	43152	83592	666	1763
1.75	832	1609	4.5	7.8	1398	3402	8.8	20	6377	18871	82	267
2	1088	2507	6.2	13	4432	13400	27	91	17778	45829	157	434
3	230	610	1.4	3.8	2952	11967	16	55	3666	10530	29	76
av.	811	3947	4.6	21	3711	44137	27	371	22907	289781	222	1849

	70				80				90			
	nodes		*cpu(s)*		*nodes*		*cpu(s)*		*nodes*		*cpu(s)*	
R	*av.*	*max*	*av.*	*max*	*av.*	*max*	*av.*	*max*	*av.*	*max*	*av.*	*max*
0.2	248	601	5.1	12	269	671	7.8	16	2077	5724	57	148
0.4	6809	29234	85	351	2412	5117	45	98	9408	25276	202	511
0.6	9974	20879	93	168	7524	21002	105	306	64034	243740	809	2839
0.8	6847	9374	54	78	98581	406075	875	3274	281407	689011	2271	4958
1	106768	258494	948	2530	271072	525576	2248	**(1)**	127096	274921	1309	2683
1.25	124119	516662	1416	**(1)**	186916	474430	2892	**(2)**	185616	419033	2668	**(2)**
1.5	103170	349804	1335	3373	111164	230834	1669	3609	193113	433350	2865	**(2)**
1.75	21584	41391	249	490	153938	321038	1707	3268	257589	470730	3617	**(3)**
2	12900	40600	163	510	131216	381472	1750	**(1)**	213096	433644	2524	**(1)**
3	5338	9800	86	190	18140	30954	332	614	53406	201260	872	3767
av.	39776	516662	443	**(1)**	98123	525576	1163	**(4)**	138684	689011	1719	**(8)**

Table 4 $1, NI|r_i|\sum w_iC_i$: main results

computation time. The time limit and the number of generated nodes at this time limit are then used for the statistics. For $1, NI|r_i|\sum C_i$, all instances are solved within the time limit, apart from one instance of 90 jobs. This is one of the cases where the wrong branching strategy was chosen. This instance can in fact be solved in 1637 seconds and 254477 nodes if the F strategy is chosen instead of strategy L. The instances of the $1, NI|r_i|\sum w_iC_i$ are much harder to solve. Thus, an instance of 70 jobs is not solved within the time limit. Here, if the strategy L had been chosen instead of F, the instance could have been solved in 405 seconds and 70021 nodes. Note that two instances of 80 jobs are not solved within the time limit for any strategy.

9 Conclusion

In this article we have studied the minimization of a regular criterion when jobs with release dates are to be processed by a machine subject to the non-idling constraint. A branch and bound procedure to solve this problem has been presented. Several properties and techniques have been described for this problem. An application to $1, NI|r_i|\sum w_iC_i$ and $1, NI|r_i|\sum C_i$ has been proposed, and experimental results have been provided. To our knowledge, this is the first attempt to solve these problems exactly. Experimental results show the efficiency of the proposed techniques used with our method as well with the "Reduce-To-The-Opt" algorithm of Wolf (30). As regards future work, a study of different branching schemes and strategies might be undertaken. As shown by the experimental results, our method for selecting the best branching strategy sometimes fails. Using strategies fixing the start time of sequences as the "Reduce-To-The-Opt"

algorithm seems to be competitive and could be more deeply studied. Moreover, a dynamic programming approach could also be investigated, since some of the properties of the problem would tend to suggest this approach.

Acknowledgments

The work presented in this article has been partially supported by LMCO ANR project. The author would like to thank Jacques Carlier for enlightening discussions on this problem. The author is also very grateful to the anonymous referees for their careful review and for their suggestions regarding the improvement of this article.

*Bibliography

1. K.R. Baker. *Introduction to Sequencing and Scheduling*. John Wiley and Sons, 1974.
2. Ph. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling, Applying Constraint Programming to Scheduling Problems*, volume 39 of *International Series in Operations Research and Management Science*. Kluwer, 2001.
3. 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.
4. L. Bianco and S. Ricciardelli. Scheduling of a single machine to minimize total weighted completion time subject to release dates. *Naval Research Logistics Quarterly*, 29:151–167, 1982.
5. P. Brucker. *Scheduling Algorithms*. Springer Lehrbuch, 1995.
6. J. Carlier. Ordonnancements à contraintes disjonctives. *RAIRO*, 12:333–351, 1978.
7. J. Carlier, F. Hermès, A. Moukrim and K. Ghedira. Exact resolution of the one-machine sequencing problem with no machine idle time. *submitted*, 2009.
8. S. Chand, R. Traub, and R. Uzsoy. Single-machine scheduling with dynamic arrivals: Decomposition results and an improved algorithm. *Naval Research Logistics*, 43:709–716, 1996.
9. R. Chandra. On $n/1/\overline{F}$ dynamic determistic systems. *Naval Research Logistics*, 26:537–544, 1979.
10. P. Chrétienne. On single-machine scheduling without intermediate delays. *Discrete Applied Mathematics*, 156:2543–2550, 2008.
11. C. Chu. A branch and bound algorithm to minimize total flow time with unequal release dates. *Naval Research Logistics*, 39:859–875, 1992.
12. C. Chu. A branch and bound algorithm to minimize total tardiness with different release dates. *Naval Research Logistics*, 39:265–283, 1992.
13. R.W. Conway, W.L. Maxwell, and L.W. Miller. *Theory of Scheduling*. Addison-Wesley, Reading, 1967.
14. F. Della-Croce and V. T'kindt. Improving the preemptive bound for the one-machine dynamic total completion time scheduling problem. *Operations Research Letters*, 31:142–148, 2003.
15. D.S. Deogun. On scheduling with ready times to minimize mean flow time. *Computer Journal*, 26:320–328, 1983.
16. M.I. Dessouky and D.S. Deogun. Sequencing jobs with unequal ready times to minimize mean flow time. *SIAM Journal on Computing*, 10:192–202, 1981.
17. A. Hariri and C. 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. *Ordonnancer une machine pour minimiser la somme des coûts – The one machine total cost problem*. PhD thesis, Université de Technologie de Compiègne, France, 2002.
19. A. Jouglet, Ph. Baptiste, and J. Carlier. Branch-and-Bound Algorithms for Total Weighted Tardiness, pages In *Handbook of scheduling : algorithms, models, and performance analysis*, J. Y-T Leung , Chapman & Hall / CRC edition, 13:1–21, 2004.
20. A. Jouglet, D. Savourey, J. Carlier, and Ph. Baptiste. Dominance-based heuristics for one-machine total cost scheduling problems. *European Journal of Operational Research*, 184(3):879–899, 2008.
21. J. Labetoulle, E.L. Lawler, J. Lenstra, and Kan A.R. Preemptive scheduling of uniform machines subject to release dates. *Progress in combinatorial optimization (Waterloo, Ontario, 1982), Academic Press*, pages 245–261, 1984.
22. J. Lenstra, A.R. Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.
23. O. Lhomme. Consistency techniques for numeric CSPs. *Thirteenth International Joint Conference on Artificial Intelligence, Chambéry, France*, 1993.
24. G. Rinaldi and A. Sassano. On a job scheduling problem with different ready times: Some properties and a new algorithm to determine the optimal solution. Rapporto dell'Ist. di Automatica dell'Universita di Roma e del C.S.S.C.C.A.-C.N.R.R, Report R.77-24.

25. A.H.G. Rinnooy Kan. *Machine sequencing problem: classification, complexity and computation.* Nijhoff. The Hague, 1976.
26. L.E. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, pages 6(3):687–690, 1968.
27. W.E. Smith. Various optimizers for single stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.
28. J.M.S. Valente and R.A.F.S Alves. An exact approach to early/tardy scheduling with release dates. *Computers and Operations Research*, 32:2905–2917, 2006.
29. G. Verfaillie and T. Schiex. Maintien de solutions dans les problèmes dynamiques de satisfaction de contraintes : bilan de quelques approches. *Revue d'Intelligence Artificielle*, 9(3), 1995.
30. A. Wolf. Reduce-To-The-Opt - a specialized search algorithm for contiguous task scheduling. In K.R. Apt, F. Fages, F. Rossi, P. Szeredi, and J. Váncza (Eds), *Recent Advances in Constraints*, Lecture Notes in Artificial Intelligence, 3010:223–232, Springer Verlag, 2004.