A Branch-and-Bound Procedure to Minimize Total Tardiness on One Machine with Arbitrary Release Dates *

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

Abstract

In this paper, we present a Branch-and-Bound procedure to minimize total tardiness on one machine with arbitrary release dates. We introduce new lower bounds and we generalize some well-known dominance properties. Our procedure handles instances as large as 500 jobs although some 60 jobs instances remain open. Computational results show that the proposed approach outperforms the best known procedures.

Key words: One Machine Scheduling, Total Tardiness

*This work has been partially financed by ILOG S.A., under research contract ILOG/GRADIENT 201 (2000-2001).

[1]IBM Watson Research Center, Math. Sciences, Office 35-214, PO Box 218, Yorktown Heights, NY 10598, USA

[2]HeuDiaSyC, UMR CNRS 6599, Université de Technologie de Compiègne, Centre de Recherches de Royallieu, BP 20529, 60205 Compiègne Cedex, France

1 Introduction, General Framework

In this paper we consider the scheduling situation where n jobs $J_1, \ldots, J_n$ have to be processed by a single machine and where the objective is to minimize total tardiness. Associated with each job J_i, are a processing time p_i, a due date d_i, and a release date r_i. 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. The tardiness of a job J_i is defined as $T_i = \max(0, C_i - d_i)$, where C_i is the completion time of J_i. The problem is to find a feasible schedule with minimum total tardiness $\sum T_i$. The problem, denoted as $1|r_i|\sum T_i$, is known to be NP-hard in the strong sense [15].

A lot of research has been carried on the problem with equal release dates $1||\sum T_i$. Powerful dominance rules have been introduced by Emmons [10]. Lawler [12] has proposed a dynamic programming algorithm that solves the problem in pseudo-polynomial time. This algorithm has been extended for scheduling serial batching machines by Baptiste and Jouglet [3]. Finally, Du and Leung have shown that the problem is NP-Hard [9]. Most of the exact methods for solving $1||\sum T_i$ strongly rely on Emmons' dominance rules. Potts and Van Wassenhove [14], Chang *et al.* [6] and Szwarc *et al.* [16], have developed Branch-and-Bound methods using the Emmons rules coupled with the decomposition rule of Lawler [12] together with some other elimination rules. The best results have been obtained by Szwarc, Della Croce and Grosso [16] with a Branch-and-Bound method that efficiently handles instances with up to 500 jobs.

There are less results on the problem with arbitrary release dates. Chu and Portmann [7] have introduced a sufficient condition for local optimality which allows to build a dominant subset of schedules. Chu [8] has also proposed a Branch-and-Bound method using some efficient dominance rules. This method handles instances with up to 30 jobs for the hardest instances and with up to 230 jobs for the easiest ones.

The aim of this paper is to propose an efficient Branch-and-Bound method with constraint propagation to solve the problem with arbitrary release dates.

In the remaining parts of this introduction, we describe the general framework of our Branch-and-Bound procedure. The several "ingredients" that make it efficient will be described in the remaining sections.

In the following, a **time window** is associated with each job J_i; it represents the time interval within each job can be scheduled. Before starting the search, the time window of J_i is initially set to $[r_i, M)$, where r_i is the release date of the job and where M is a large value such as the largest release date plus the sum of the processing times. During the Branch-and-Bound, the time windows are tightened due to decisions taken at each node of the search tree together with constraint propagation and dominance rules. To keep it simple, we will say that the lower bound of the time window is the **release date** r_i of J_i (while this value can be larger than the initial release date). The upper bound of the time window δ_i will be referred to as the **deadline** of J_i.

In Constraint Programming terms, a variable C_i representing the completion time of J_i is associated with each job. Its domain is $[r_i + p_i, \delta_i)$. The criterion is an **additional variable** $\bar{T}$ that is constrained to be equal to $\sum_i \max(0, C_i - d_i)$. Arc-B-Consistency (see for instance [13]) is used to propagate this constraint. It ensures that when a schedule has been found, the value of $\bar{T}$ is actually the total tardiness of the schedule. To find an optimal solution, we **solve successive variants of the decision problem**. At each iteration, we try to improve the best known solution and thus, we add an additional constraint stating that $\bar{T}$ is lower than or equal to the best solution minus 1. It now remains to show how to solve the decision variant of the problem.

We rely on the **edge-finding** branching scheme (see for instance [5]). Rather than searching for the starting times of jobs, we look for a **sequence** of jobs. This sequence is built both from the beginning and from the end of the schedule. Throughout the search tree, we dynamically maintain several sets of jobs that represent the current state of the schedule (see Figure 1).

- P is the sequence of the jobs scheduled at the beginning,

- Q is the sequence of the jobs scheduled at the end,
- NS is the set of unscheduled jobs that have to be sequenced between P and Q,
- $PF \subseteq NS$ (Possible First) is the set of jobs which can be scheduled immediately after P
- and $PL \subseteq NS$ (Possible Last) is the set of jobs which can be scheduled immediately before Q.

At each node of the search tree, a job J_i is chosen among those in PF and it is scheduled immediately after P. Upon backtracking, this job is removed from PF. The **heuristic** used to select J_i comes from [7, 8]: Among jobs of PF with minimal release date, select the job that minimizes the function $\max(r_i + p_i, d_i)$. Of course, if NS is empty then a solution has been found and we can iterate to the next decision problem. If $NS \neq \emptyset$ while PF or PL is empty then a backtrack occurs. Moreover, several propagation rules relying on jobs time-windows, are used to update and adjust these sets. These rules, known as edge-finding [5, 2], are also able to adjust the time-windows according to the machine constraint.

Due to our branching scheme, jobs are sequenced from left to right so it may happen that at some node of the search tree, all jobs of NS have the same release date (the completion time of the last job in P). In such a case, to improve the behavior of the Branch-and-Bound method, we apply the dynamic programming algorithm of Lawler [12] to optimally complete the schedule.

Such a Branch-and-Bound procedure is very easy to implement on top of a constraint-based scheduling system such as ILOG SCHEDULER. Unfortunately, it does not perform well since no specific technique has been used in the above formulation to solve the total tardiness problem. In the following sections, we introduce dominance properties (Section 2), lower-bounds (Section 3) and constraint propagation techniques (Section 4) to improve

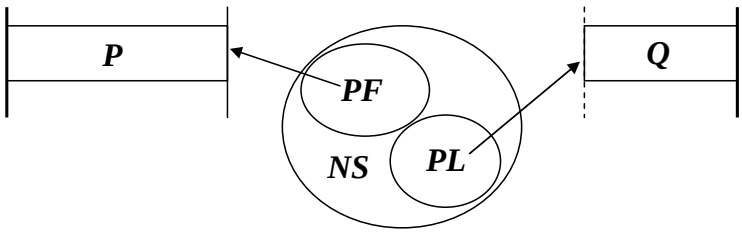

Figure 1: A Node of the Search Tree.

the behavior of the Branch-and-Bound method. Finally, our computational results are reported in Section 5.

2 Dominance Properties

A dominance rule is a constraint that can be added to the initial problem without changing the value of the optimum, *i.e.*, there is at least one optimal solution of the problem for which the dominance holds. Dominance rules can be of prime interest since they can be used to reduce the search space. However they have to be used with care since the optimum can be missed if conflicting dominance rules are combined.

In this section, we present a generalization of Emmons dominance rules that are valid either if preemption is allowed or if jobs have identical processing times. We also propose some dominance rules relying on the sequence P. Finally, we introduce an Intelligent Backtracking scheme.

2.1 Emmons Rules

We recall Emmons Rules and generalize them to take into account release dates. Unfortunately, these rules are valid if preemption is allowed (or, for some of them, if jobs have identical processing times). Nevertheless, we will see in Section 3 that these rules can be used in a pre-processing phase before computing a preemptive lower-bound of the problem.

2.1.1 Initial Emmons Rules

Emmons [10] has proposed a set of dominance rules for the special case where release dates are equal ($1||\sum T_i$). These rules allow us to deduce some precedence relations between jobs. Following Emmons notation, A_i and B_i are the sets of jobs that have to be scheduled, according to the dominance rules, respectively after and before J_i. In the following, we say that J_i precedes J_k when there is an optimal schedule in which J_i precedes J_k and for which all previously mentioned dominance properties hold.

Emmons Rule 1. $\forall i, k (i \neq k)$, *if* $p_i \leq p_k$ *and* $d_i \leq \max(\sum_{J_j \in B_k} p_j + p_k, d_k)$ *then* J_i *precedes* J_k *(*$J_i \in B_k$ *and* $J_k \in A_i$*).*

Emmons Rule 2. $\forall i, k (i \neq k)$, *if* $p_i \leq p_k$ *and* $d_i > \max(\sum_{J_j \in B_k} p_j + p_k, d_k)$ *and* $d_i + p_i \geq \sum_{J_j \notin A_k} p_j$, *then* J_k *precedes* J_i.

Emmons Rule 3. $\forall i, k (i \neq k)$, *if* $p_i \leq p_k$ *and* $d_k \geq \sum_{J_j \notin A_i} p_j$, *then* J_i *precedes* J_k.

2.1.2 Generalized Emmons Rules $(1|r_i, pmtn, \sum T_i)$

Our aim is to generalize Emmons rules to the situation where we have arbitrary release dates. Such a generalization is relatively easy to do if we relax the non-preemption constraint. We will see that the resulting rules can be used to tighten the lower bound of the non preemptive problem (Section 3.4).

In the preemptive case, J_i is said to precede J_k if and only if J_k starts after the end of J_i. As in Section 2.1.1, A_i and B_i respectively denote the set of jobs that are known to execute after and before J_i.

Note that active schedules are dominant, *i.e.*, we only consider schedules in which jobs or pieces of jobs cannot be scheduled earlier without delaying another job. It is easy to see that all active schedules have exactly the same completion time C_{max}. To compute this value, we can build the schedule where jobs are scheduled in non decreasing order of release dates. Now we can tighten the deadlines since J_i since it cannot be completed after $C_{max} - \sum_{J_j \in A_i} p_j$.

In the following, we note $C_{max}(E)$ the completion time of active schedules of a subset E of jobs (other jobs are not considered). As mentioned previously, all active schedules have the same completion time and it can be computed in polynomial time. $C_{max}(E)$ is a lower bound of the maximal completion time of the jobs of E in any schedule of $\{J_1, \ldots, J_n\}$. If B_i, as in Section 2.1.1, is a set of jobs that have to be processed before J_i then J_i cannot start before $C_{max}(B_i)$, *i.e.*, the **release date** r_i can be **adjusted** to $\max(r_i, C_{max}(B_i))$.

First, we make the following remark which allows us to compare the values of the tardiness of a job J_i in two schedules S and S'.

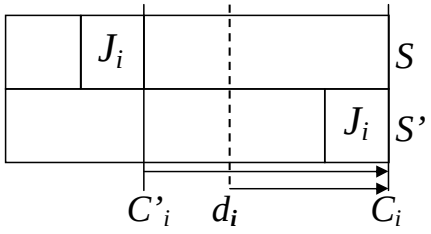

Figure 2: Case where $C'_i < C_i$ and $d_i \leq C_i$.

Remark 1. *Let C_i and C'_i be the completion times of J_i on two preemptive schedules S and S' and let T_i and T'_i be respectively the tardiness of J_i in S and S'.*

- *If $C'_i < C_i$ and $d_i \leq C_i$ then $T'_i - T_i = \max(C'_i, d_i) - C_i \leq 0$ (see Figure 2).*
- *If $d_i \geq \max(C_i, C'_i)$ then J_i is on time in S and in S' and $T'_i - T_i = 0$.*

Generalized Emmons Rule 1. *Let J_i and J_k be two jobs such that $r_i \leq r_k$, $p_i \leq p_k$ and $d_i \leq \max(r_k + p_k, d_k)$, then J_i precedes J_k ($J_i \in B_k$ and $J_k \in A_i$).*

Proof. Consider a schedule S in which job J_i and job J_k satisfy the assumptions. First, assume that job J_k is completed before job J_i ($C_k < C_i$). Let

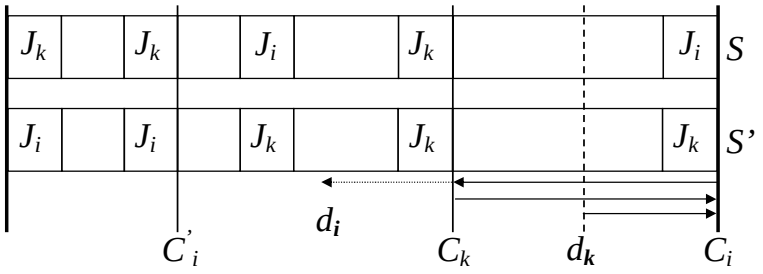

Figure 3: Generalized Emmons Rule 1.

us reschedule the pieces of J_i and J_k such that all pieces of J_i are completed before the pieces of J_k (see Figure 3). Note that the exchange is valid since r_i is lower than or equal to r_k. We show that this exchange does not increase total tardiness. Let S' be this new schedule and let C'_i be the completion time of J_i in S'. In S', J_k is completed at time C_i. If $C_i \leq d_i$, the jobs J_i and J_k are on time in the two schedules S and S' and the exchange has no effect. From now, we suppose that $d_i < C_i$.

Since the completion times of all other jobs remain the same, the difference between the total tardiness of S' and of S is exactly $\Delta = T'_i - T_i + T'_k - T_k$. Following Remark 1, $T'_i - T_i = \max(C'_i, d_i) - C_i$ and $T'_k - T_k = C'_k - \max(C_k, d_k) = C_i - \max(C_k, d_k)$. So, $\Delta = \max(C'_i, d_i) - \max(C_k, d_k)$. Note that all pieces of J_i are completed before C_k in S' and since $p_i \leq p_k$, we have $C'_i \leq C_k$ and $\Delta \leq \max(C_k, d_i) - \max(C_k, d_k)$.

Now consider two cases:

- If $\max(r_k + p_k, d_k) = d_k$ then $d_i \leq d_k$. Together with $C'_i \leq C_k$, this leads to $\Delta \leq 0$.
- If $\max(r_k + p_k, d_k) = r_k + p_k$ then $d_i \leq r_k + p_k \leq C_k$ and $d_k \leq r_k + p_k \leq C_k$. This leads to $\Delta \leq C_k - C_k = 0$.

Now assume, that job J_i is completed before job J_k. Rescheduling the pieces of the two jobs, such that all pieces of J_i are completed before the pieces of J_k, can only decrease the tardiness of J_i, and leave unchanged the tardiness

of J_k.
In all cases, the exchange does not increase total tardiness. □

As shown in Figure 4, the Generalized Emmons Rule 1, does not hold in the non-preemptive case. Indeed, J_1 would have to be completed before J_2, which is not true in the non-preemptive case.

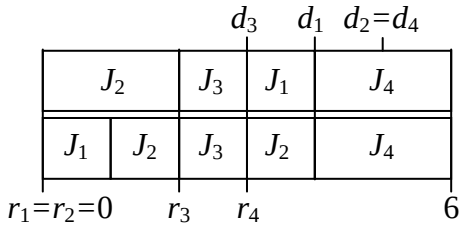

Figure 4: An Optimal Non-Preemptive Schedule and an Optimal Preemptive Schedule.

Generalized Emmons Rule 2. *Let J_i and J_k be two jobs such that $r_i \leq r_k$, $d_i \leq d_k$ and $\delta_i \leq d_k + p_k$, then J_i precedes J_k*

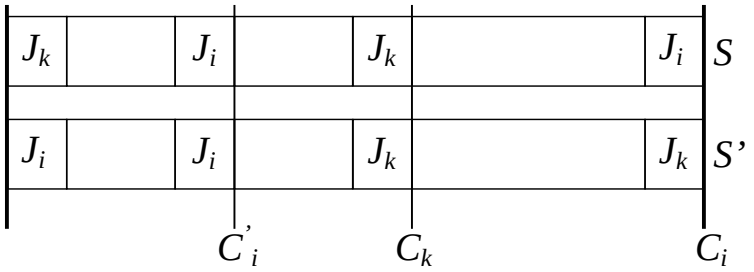

Figure 5: Generalized Emmons Rule 2.

Proof. Consider a schedule S in which job J_i and job J_k satisfy the assumptions. First, assume that job J_k is completed before job J_i ($C_k < C_i$). Moreover $C_i \leq \delta_i$. Let us reschedule the pieces of J_i and J_k such that all pieces of J_i are completed before the pieces of J_k. Note that the exchange is valid since r_i is lower than or equal to r_k (see Figure 5). We show that this

exchange does not increase total tardiness. Let S' this new schedule and let C'_i, the completion time of J_i in S'. In S', J_k is completed at time C_i. If $C_i \leq d_i$ the jobs J_i and J_k are on time in the two schedules S and S' and the exchange has no effect. From now, we suppose that $d_i < C_i$.
Since the completion times of all other jobs remain the same, the difference between the total tardiness of S' and of S is exactly $\Delta = T'_i - T_i + T'_k - T_k$. Following Remark 1, $T'_i - T_i = \max(C'_i, d_i) - C_i$ and $T'_k - T_k = C'_k - \max(C_k, d_k) = C_i - \max(C_k, d_k)$. Note that all pieces of J_i are completed before C_k in S' and since $p_i \leq p_k$, we have $C'_i \leq C_k$.
We have $d_k + p_k \geq \delta_i \geq C_i$, thus $d_k \geq C_i - p_k$. All pieces of J_k are scheduled between C'_i and C_i so $C'_i \leq C_i - p_k$ and $T'_i - T_i = \max(C'_i, d_i) - C_i \leq \max(C_i - p_k, d_i) - C_i$. Thus $\Delta = T'_i - T_i + T'_k - T_k \leq \max(C_i - p_k, d_i) - \max(C_k, d_k)$. We have $d_i \leq d_k$ and $C_i - p_k \leq d_k$ then $\Delta \leq 0$.
Now assume that J_i is completed before job J_k. Rescheduling the pieces of the two jobs, such that all pieces of J_i are completed before the pieces of J_k, can only decrease the tardiness of J_i, and leave unchanged the tardiness of J_k.
In all cases, the exchange does not increase total tardiness. □

Generalized Emmons Rule 3. *Let J_i and J_k be two jobs such that $r_i \leq r_k$ and $\delta_i \leq d_k$, then J_i precedes J_k.*

Proof. Consider a schedule S in which job J_i and job J_k satisfy the assumptions. First, assume that job J_k is completed before job J_i ($C_k < C_i$) and

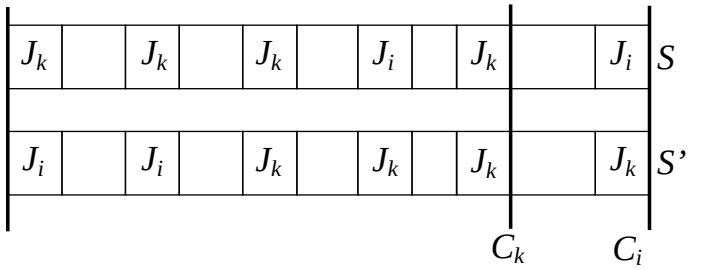

Figure 6: Generalized Emmons Rule 3.

that $C_i \leq \delta_i$. Let us reschedule the pieces of J_i and J_k such that all pieces of J_i are completed before the pieces of J_k. Note that the exchange is valid since r_i is lower than or equal to r_k (see Figure 6). We show that this exchange does not increase total tardiness. Let S' this new schedule and let C'_i, the completion time of J_i in S'. In S', J_k is completed at time C_i. If $C_i \leq d_i$ the jobs J_i and J_k are on time in the two schedules S and S' and the exchange has no effect. From now, we suppose that $d_i < C_i$.
Since the completion times of all other jobs remain the same, the difference between the total tardiness of S' and of S is exactly $\Delta = T'_i - T_i + T'_k - T_k$. Following Remark 1, $T'_i - T_i = \max(C'_i, d_i) - C_i \leq 0$ and $T'_k - T_k = C'_k - \max(C_k, d_k) = C_i - \max(C_k, d_k)$. We have $C_k \leq C_i \leq \delta_i \leq d_k$ then J_k is on time in S and in S'. Thus $T'_k - T_k = 0$ and $\Delta \leq 0$.
Now assume that J_i is completed before job J_k. Rescheduling the pieces of the two jobs, such that all pieces of J_i are completed before the pieces of J_k, can only decrease the tardiness of J_i, and leave unchanged the tardiness of J_k.
In all cases, the exchange does not increase total tardiness. □

2.1.3 Combining Generalized Emmons Rules

As mentioned previously, dominance properties cannot always be combined. Fortunately, in our case, the Generalized Emmons Rules are compatible, which was already the case for $1||\sum T_i$ as noticed by Emmons [10].

We can use these dominance rules one after the other to adjust the data of an instance: if a set B_i of jobs is proved to precede J_i according to the Generalized Emmons Rules, it is possible to adjust r_i to $r_i = \max(r_i, C_{max}(B_i))$. By using iteratively these rules, we are sure, that at the end of each iteration, the adjusted instance has the same optimal total tardiness than the original instance, and that we have not introduce any inconsistencies.

We consider jobs one after the other and each time we perform adjustments. Now, we can prove that we do not introduce any inconsistencies. Indeed, suppose that at the current iteration, we have for each job J_i, a set

B_i of jobs which are proven to precede J_i, and a set A_i of jobs which are proven to follow J_i. Suppose too that there is no contradiction at this time and that all adjustments are made considering these sets. It means that we have $r_j < r_i$, for any $J_j \in B_i$ and $r_i < r_j$, for any $J_j \in A_i$. Suppose now, we want to refine the adjustments for a job J_k. Let $J_i \notin B_k$ be a job which has not yet been proved to precede J_k, but for which, one of the Generalized Emmons Rule has proven that J_i precedes J_k. This implies that $r_i \leq r_k$. According to this rule, we can introduce a contradiction only if there exists a job J_m such that: J_k precedes J_m and J_m precedes J_i. Because we have made all adjustments implied by the precedence relations already established, we have: J_k precedes J_m implies $r_k < r_m$, and J_m precedes J_i implies $r_m < r_i$, then $r_k < r_m < r_i$, which contradicts the initial assumption $r_i \leq r_k$. Consequently, we will not introduce any inconsistency according to the precedence relations already established.

Each rule can be implemented in $O(n^2)$. The jobs are sorted in non-decreasing order of release dates in a heap structure, which runs in $O(n \log n)$. For each job J_i, we consider the set of jobs which are completed before J_i according to the Generalized Emmons Rules. Computing the earliest completion time of this set and adjusting r_i runs in $O(n)$, since the jobs are already sorted in non-decreasing order of release dates. The heap structure is maintained in $0(\log n)$. Therefore, we have an overall time complexity of $O(n)$ for each job. New precedence dominances can be found and the rules might have to be applied again. Each job can be adjusted at most $n - 1$ times. Hence, in the worst case the whole propagation is in $O(n^3)$.

2.1.4 Equal Length Jobs

We come back to the initial non-preemptive problem with release dates. We prove that, considering two jobs which have the same processing time, the first Emmons Rule is valid. Note that, contrary to the Generalized Emmons Rules, this dominance property is valid even in the non-preemptive case.

Dominance Rule 1. *Let J_i and J_k be two jobs such that $p_i = p_k$. If $r_i \leq r_k$ and $d_i \leq \max(r_k + p_k, d_k)$, then there exists an optimal schedule in which J_i precedes J_k.*

Proof. Consider a schedule S in which job J_i and job J_k satisfy the assumptions and in which job J_k is completed before job J_i. Let us exchange J_i and J_k in S. Note that the exchange is valid since r_i is lower than or equal to the earliest start time of J_k. We have $p_i = p_k$, then the completion time of the other jobs do not change. This exchange does not increase total tardiness as it has been proved for the Generalized Emmons Rule 1. □

2.2 Removing Dominated Sequences

Several dominance properties have been introduced in [7, 8]. These rules focus on the jobs in NS plus the last job of P. They determine that some precedence constraints can be added. Such constraints allow us to adjust release dates and to filter PF. All these rules are used in our Branch-and-Bound procedure. On top of this, we also consider dominance properties that take into account the **complete sequence** P. Informally speaking, our most basic rule states that if the current sequence P can be "improved", then it is dominated and we can backtrack.

In the following, let $C_{\max}(P)$ and $T(P)$ denote the completion time and the total tardiness associated with the current sequence P. Now consider a permutation P' of P of and let us examine under which condition P' is "as good as" than P.

- If $C_{\max}(P') \leq C_{\max}(P)$ and $T(P') \leq T(P)$, then we can replace P by P' in any feasible schedule so P' is at least as good as P.
- If $C_{\max}(P') > C_{\max}(P)$, then if we replace P by P' in a feasible schedule, all jobs in $NS \cup Q$ have to be shifted of at most $C_{\max}(P') - C_{\max}(P)$ time units. So, the additional cost for jobs in $NS \cup Q$ is at most $(|NS| + |Q|)(C_{\max}(P') - C_{\max}(P))$. Hence, P' is at least as good as P if $T(P') + (|NS| + |Q|)(C_{\max}(P') - C_{\max}(P)) \leq T(P)$.

P' is better than P if P' is at least as good as P and if either (1) P is not at least as good as P' (2) or if P' is lexicographically smaller than P. When P' is better than P, the current sequence is dominated and we can backtrack. To compare two sequences, we just have to build the schedules associated with them and this can be done in linear time.

2.2.1 Enumeration of the k Last Jobs

We can enumerate all permutations P' of P and test if it is better than P. Since the number of alternative sequences is exponential, we only consider the permutations P' that are identical to P except for the k last jobs (where k is an arbitrary value). When k is large, we have a great reduction of the search space but this takes a lot of time. Experimentally, $k = 6$ seems to be a good trade-off between the reduction of the search tree and the time spent in the enumeration.

2.2.2 Removing Possible Firsts

We propose a simple technique to detect that a job $J_i \in PF$ cannot be actually scheduled just after P and thus can be removed from PF. We note $P|J_i$ the sequence where J_i is scheduled immediately after P. If there is a permutation π of $P|J_i$ that is "better" than $P|J_i$ then scheduling J_i first after P leads to a dominated schedule and thus we can remove J_i from PF. This time we only enumerate the permutations π that are obtained from $P|J_i$ either by inserting J_i somewhere inside P or by exchanging J_i with another job of $P|J_i$. Since there are $O(n)$ such permutations and since comparing two sequences can be done in linear time, the algorithm runs in $O(n^2)$ for a given job J_i.

2.2.3 Adjusting Deadlines

Consider a job $J_i \in NS$ and assume that the current node of the search tree can be extended to a feasible optimal schedule where J_i is completed

at some time point C_i. Let $P|\lambda|J_i|\mu|Q$ be the sequence of the jobs in the feasible optimal schedule and let us modify this sequence by removing J_i and inserting it somewhere in P. The new sequence is $P'|J_i|P''|\lambda|\mu|Q$, where $P'|J_i|P''$ is the sequence derived from P after the insertion of J_i.

In the following, we assume that $r_i \leq C_{\max}(P')$. Under this hypothesis, it is easy to see that $C_{\max}(P'|J_i|P''|\lambda) \leq C_{\max}(P|\lambda|J_i)$ and thus the **tardiness of the jobs in $\mu|Q$** has not been increased. On top of that, the jobs in λ have been shifted of at most $C_{\max}(P'|J_i|P'') - C_{\max}(P)$ time units. Thus, the **total tardiness of the jobs in λ** has been increased of at most $|\lambda|(C_{\max}(P'|J_i|P'') - C_{\max}(P)) \leq (|NS| - 1)(C_{\max}(P'|J_i|P'') - C_{\max}(P))$. Finally, the **total tardiness of the jobs in $P \cup \{J_i\}$** has been increased of $T(P'|J_i|P'') - (T(P) + \max(0, C_i - d_i))$. Consequently, the total tardiness has been increased of at most

$$(|NS|-1)(C_{\max}(P'|J_i|P'')-C_{\max}(P))+T(P'|J_i|P'')-T(P)-\max(0,C_i-d_i).$$

Since the initial schedule is optimal, the above expression is positive and thus,

$$C_i < d_i + (|NS| - 1)(C_{\max}(P'|J_i|P'') - C_{\max}(P)) + T(P'|J_i|P'') - T(P).$$

In other words, the deadline δ_i of J_i can be adjusted to

$$\min(\delta_i, d_i - 1 + (|NS| - 1)(C_{\max}(P'|J_i|P'') - C_{\max}(P)) + T(P'|J_i|P'') - T(P)).$$

Since there are $O(n)$ sequences $P'|J_i|P''$ (with $r_i \leq C_{\max}(P')$) that can be derived from P and since the completion time and the total tardiness of a sequence can be computed in linear time, all adjustments of related to J_i can be computed in $O(n^2)$.

2.3 Intelligent Backtracking

Intelligent Backtracking techniques "record", throughout the search tree, some situations that do not lead to a feasible (non-dominated) solution. Each

time such a situation is encountered again then a backtrack is immediately triggered.

Given a node ν of the search tree, let P^ν, Q^ν, NS^ν, PF^ν, PL^ν, $r_1^\nu, ..., r_n^\nu$, $\delta_1^\nu, ..., \delta_n^\nu$ be the "values" taken by P, Q, NS, PF, PL, $r_1, ..., r_n$, $\delta_1, ..., \delta_n$ at node ν. Whenever a backtrack occurs, the node ν is stored in a "no-good" list. Now assume that we encounter another node μ such that (1) P^ν is a permutation of P^μ, (2) $C_{\max}(P^\nu) \leq C_{\max}(P^\mu)$, (3) $T(P^\nu) \leq T(P^\mu)$, (4) $PF(\mu) \subseteq PF(\nu)$, (5) $PL(\mu) \subseteq PL(\nu)$, (6) $Q^\mu = Q^\nu$, (7) $\forall i, r_i^\mu \geq r_i^\nu$ and (8) $\forall i, \delta_i^\mu \leq \delta_i^\nu$. Then μ is dominated by ν and we can immediately backtrack.

Conditions (2) and (3) can be slightly weakened to take into account the case where $C_{\max}(P^\nu) > C_{\max}(P^\mu)$. Indeed, we know that $n - |P^\mu| = |NS^\mu|$ jobs remain to be scheduled after P^μ and decreasing the completion time $C_{\max}(P^\mu)$ can decrease the tardiness of each of these jobs of at most $C_{\max}(P^\nu) - C_{\max}(P^\mu)$. Thus, if

$$T(P^\nu) \leq T(P^\mu) - (n - |P^\mu|)(C_{\max}(P^\nu) - C_{\max}(P^\mu)),$$

and if all other conditions hold, μ is dominated by ν and we can immediately backtrack.

3 Lower Bounds

Relaxing non-preemption is a standard technique to obtain lower bounds for non-preemptive scheduling problems. Unfortunately, the problem with no release dates (for which the preemption is not useful) is already NP-Hard, so some additional constraints have to be relaxed to obtain a lower bound in polynomial time. From now on, assume that jobs are sorted in **non-decreasing order of due dates**. First, Chu's lower bound is recalled then, a new lower bound is described and finally, we show how the Generalized Emmons Rules can be used to improve this lower bound.

3.1 Chu's Lower Bound

Chu [8] has introduced an $O(n \log n)$ lower bound, $\boldsymbol{lb_{Chu}}$, that can be computed as follows. Preemption is relaxed and jobs are scheduled according to the SRPT (Shortest Remaining Processing Time) rule. Each time a job becomes available or is completed, a job with the shortest remaining processing time among the available and uncompleted jobs is scheduled (see Table 1 and Figure 7). The computation of the lower bound is based on Proposition 1 ([8]).

J_i	r_i	p_i	d_i
J_1	0	5	5
J_2	1	4	6
J_3	3	1	8

Table 1: An Instance of $1|r_i|\sum T_i$.

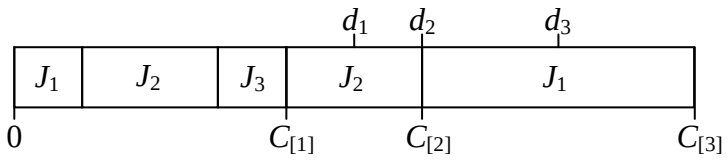

Figure 7: An SRPT Schedule of the Instance of Table 1.

Proposition 1. *Let $[i]$ denote the index of the job which is completed in the i^{th} position in the SRPT schedule. The total tardiness is at least*

$$\sum_{i=1}^{n} \max(C_{[i]} - d_i, 0).$$

3.2 A New Lower Bound

We introduce two propositions, that are valid in the preemptive case only. Proposition 2 is a weak version of the Generalized Emmons Rule 1. Proposi-

tion 3 shows that under some conditions, due dates can be exchanged without increasing the objective function.

Proposition 2. *Let J_i and J_k be two jobs such that $r_i \leq r_k$, $p_i \leq p_k$ and $d_i \leq d_k$, then there exists an optimal schedule in which J_k starts after the end of J_i.*

Proof. See proof of Generalized Emmons Rule 1. □

Proposition 3. *Let J_i and J_k be two jobs such that $r_i \leq r_k$, $p_i \leq p_k$ and $d_i > d_k$. Exchanging d_i and d_k does not increase the optimal total tardiness.*

Proof. Consider an optimal schedule S of the original instance and let C_i and C_k be the completion times of J_i and J_k in S. Let Δ be the difference between the tardiness of S before and after the exchange.

$$\Delta = \max(0, C_i - d_k) + \max(0, C_k - d_i) - \max(0, C_i - d_i) - \max(0, C_k - d_k).$$

First, assume that $C_i < C_k$ and consider the following cases.
If $d_k < d_i \leq C_i$ then

$$\Delta = (C_i - d_k) + (C_k - d_i) - (C_i - d_i) - (C_k - d_k) = 0.$$

If $d_k \leq C_i \leq d_i \leq C_k$ then

$$\Delta = (C_i - d_k) + (C_k - d_i) - 0 - (C_k - d_k) = C_i - d_i \leq 0.$$

If $C_i \leq d_k < d_i \leq C_k$ then $\Delta = 0 + (C_k - d_i) - 0 - (C_k - d_k) = d_i - d_k \leq 0$.
If $C_i \leq d_k \leq C_k \leq d_i$ then $\Delta = 0 + 0 - 0 - (C_k - d_k) \leq 0$.
If $C_i < C_k \leq d_k < d_i$ then $\Delta = 0$.
If $d_k \leq C_i < C_k \leq d_i$ then

$$\Delta = (C_i - d_k) + 0 - 0 - (C_k - d_k) = (C_i - C_k) + (d_k - d_i) < 0.$$

Now assume that $C_k < C_i$. After the exchange of due dates, we exchange the pieces of the jobs, such that all pieces of J_i are scheduled before those of J_k.

This exchange is valid since $r_i \leq r_k$. In this new schedule, J_i is completed before or at C_k because $p_i \leq p_k$, thus the tardiness of J_i is lower than or equal to $\max(0, C_k - d_k)$. Moreover, J_k is completed at C_i and its tardiness is equal to $\max(0, C_i - d_i)$. Thus $\Delta \leq \max(0, C_i - d_i) + \max(0, C_k - d_k) - \max(0, C_i - d_i) - \max(0, C_k - d_k) \leq 0$. □

These propositions allow us to compute a lower bound $\boldsymbol{lb_1}$ thanks to the following algorithm.

At each time t, we consider $D = \{J_j / r_j \leq t \wedge p'_j > 0\}$ the set of jobs available but not completed at t (p'_j denotes the remaining processing time of J_j at time t). Let J_u be the job with the shortest remaining processing time, and let J_v be the job with the smallest due date. If $d_u = d_v$, *i.e.*, J_u has the smallest due date, according to Proposition 2, it is optimal to schedule one unit of this job. If it is not the case, according to Proposition 3, we exchange its due date with the one of J_v, the job with the smallest due date. This new instance has an optimal total tardiness lower than or equal to the optimal tardiness of the original problem. In this new problem, J_u has now the smallest due date and the smallest remaining processing time, then it is optimal to schedule one unit of this job according to Proposition 2. We increase t and we iterate until all jobs are completed.

At each time t, we build a new problem by exchanging two due dates. This new problem has an optimal total tardiness lower than or equal to the problem before. Hence, at the end of the algorithm, we obtain an optimal schedule of a problem which has a total tardiness lower than or equal to the optimal tardiness of the original problem. Therefore, it is a lower bound of the original problem. Actually, the only relevant time points are when a job becomes available or when a job is completed. And so there are at most $2n$ times t to consider.

We maintain two heaps $heap_p$ and $heap_d$ which contain respectively, the uncompleted jobs which are available at time t sorted in non-decreasing order of remaining processing times, and these jobs sorted in non-decreasing order of due dates. The insertion, the extraction and the modification of a job of

one heap costs $O(\log n)$. Getting the minimum element of one heap costs $O(1)$. We have at most n insertions (each time a job becomes available), and at most n extractions (each time a job is completed). At each time t, we have at most two modifications in $heap_p$ and in $heap_d$ when we must exchange the due dates of two jobs. Hence, our algorithm runs in $O(n \log n)$.

3.3 Comparison with Chu's Lower Bound

The schedule built by our algorithm is the same as the one built by Chu. The algorithms differ from the assignment of the due dates to the jobs. Indeed, in the algorithm of Chu, the assignment of the due dates is performed at a "global" level, whereas in our algorithm, the assignment is performed at a "local" level. We show that lb_1 is strictly better than lb_{Chu}.

Proposition 4. *The lower bound lb_1 obtained by our algorithm is stronger than Chu's lower bound lb_{Chu}.*

Proof. The sets of completion times in the schedules built by our algorithm and by Chu's are the same. The difference lies in the assignment of due dates to the jobs. Let $(d_1, \ldots, d_n)$ be the sequence of due dates sorted in non-decreasing order. This is the sequence obtained by the algorithm of Chu. Let $(d_{\sigma(1)}, \ldots, d_{\sigma(n)})$ be the sequence of due dates obtained by our algorithm. Let j be the first index such that $j \neq \sigma(j)$ and let t be the starting time of the piece of job which is completed at $C_{[j]}$. This piece is a piece of the job with the shortest remaining processing time at t. We have $d_j \in (d_{\sigma(j+1)}, \ldots, d_{\sigma(n)})$ and $d_j \leq d'_j$. Let T' be the tardiness associated with the sequence $(d_{\sigma(j+1)}, \ldots, d_{\sigma(n)})$. Let k be the integer such that $\sigma(k) = j$. Suppose we exchange the due dates $d_{\sigma(j)}$ and $d_{\sigma(k)}$ in the sequence $(d_{\sigma(j)}, \ldots, d_{\sigma(n)})$. As a result, the decrease of the total tardiness is equal to $\max(0, C_i - d_{\sigma(j)}) + \max(0, C_k - d_{\sigma(k)}) - \max(0, C_i - d_{\sigma(k)}) - \max(0, C_k - d_{\sigma(j)})$ which is non-negative as shown in the first part of the Proposition 3 because $C_i < C_k$ and $d_{\sigma(k)} \leq d_{\sigma(j)}$. Now the two sequences are identical up to index $j + 1$. We iterate until the sequences are the same. Hence $lb_{Chu} \leq lb_1$. □

We provide an example for an instance with 3 jobs, for which lb_1 is equal to 1 whereas lb_{Chu} is equal to 0 (see Table 2 and Figure 8). For this instance, lb_1 is the optimal total tardiness.

J_i	r_i	p_i	d_i
J_1	0	1	3
J_2	1	1	2
J_3	1	1	2

Table 2: Example with $lb_1 > lb_{Chu}$.

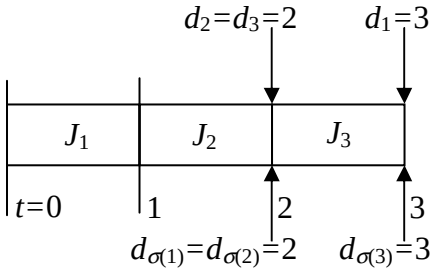

Figure 8: Assignment of Due Dates Obtained by the Algorithm Computing lb_1 (d_1, d_2, d_3) and by Chu's Algorithm $(d_{\sigma(1)}, d_{\sigma(2)}, d_{\sigma(3)})$.

3.4 Improving the Lower Bound

To improve our lower bound, we extensively use the Generalized Emmons Rules as a pre-processing step before the computation of the lower bound. The lower bound is valid in the preemptive case hence, we can apply the Generalized Emmons Rules. They allow us to tighten the release dates, which has a dramatic impact on the value of the lower bound.

Recall that, to compute our lower bound, we chronologically build a preemptive schedule (see Section 3.2). We still follow the same algorithm but, at each time point where a piece of job to schedule next has to be chosen,

we apply the Generalized Emmons Rules on the jobs that are not scheduled yet.

Since there are $O(n)$ relevant time points and since the propagation of the Generalized Emmons Rules runs in $O(n^3)$, the improved lower bound can be computed in $O(n^4)$. In practice, the propagation of the Generalized Emmons Rules is much "faster" than $O(n^3)$ and the bound is computed in a reasonable amount of time. In the following we use the notation $\boldsymbol{lb_2}$ to refer to this bound.

4 Propagation Rules

4.1 Possible and Impossible Positions

Focacci [11] has recently proposed an original approach based on Constraint Programming to compute a lower bound of $1|r_i|\sum T_i$. In this approach, each job is associated with a constrained variable identifying all possible positions (first, second, third, *etc.*) that the job can assume in a schedule. Following this idea, we present some rules that deduce that a job cannot be executed in some positions. This information allows us to adjust the release dates and to "filter" the sets PF and PL.

From now on, we assume that jobs are sorted in non-decreasing order of due dates. To simplify the presentation, we also assume that no job has been sequenced in the right part of the schedule, *i.e.*, $P = \emptyset$. If P is not empty, we can "remove" the jobs of P and apply the rules described below. Of course, the tardiness of the jobs that were in P has to be added to the lower-bounds computed below.

As in Section 3.1, $[i]$ denotes the index of the job which is completed in the i^{th} position in the SRPT schedule. We know that $C_{[i]}$ is a lower bound of the completion time of the job scheduled in i^{th} position and according to Proposition 1, $\sum_{i=1}^{n} \max(C_{[i]} - d_i, 0)$ is a lower bound of the total tardiness (each job J_i is assigned to the completion time $C_{[i]}$).

Suppose now that we want to compute a lower bound of the total tardiness under the hypothesis that J_i is scheduled in the k^{th} position. We first assign the completion time $C_{[k]}$ to J_i and we reassign all other completion times to all other jobs as follows.

- If $k < i$ the jobs $J_k, J_{k+1}, \ldots, J_{i-1}$ are assigned to $C_{[k+1]}, C_{[k+2]}, \ldots, C_{[i]}$ and the other assignments do not change.
- If $k > i$ the jobs $J_{i+1}, J_{i+2}, \ldots, J_k$ are assigned to $C_{[i]}, C_{[i+1]}, \ldots, C_{[k-1]}$ and the other assignments do not change.

Following these new assignments, we have a new lower bound. If it is greater than $\bar{T}$ (the upper bound of the objective function, as defined in Section 1), then J_i cannot be in the k^{th} position.

Now assume that we have shown that positions $1, \ldots, k$ are not possible for a job J_i then J_i cannot be completed before $C_{[k+1]}$ and cannot start before $C_{[k]}$. Hence, the release date r_i can be adjusted to $\max(r_i, C_{[k+1]} - p_i, C_{[k]})$. Moreover, if $k > 1$ then i cannot be scheduled first, *i.e.*, J_i can be removed of PF. Of course, symmetric rules hold when it is known that the last k positions are not possible for J_i.

To implement this constraint propagation rule, we just have to use the $O(n \log n)$ algorithm of Chu [8], to compute the values $C_{[1]}, C_{[2]} \ldots, C_{[n]}$. Then, for each job J_i and for each position k, the lower-bound can be recomputed in linear time thanks to the reassignment rules provided above. This leads to an overall time complexity of $O(n^3)$.

Actually, this algorithm can be improved as follows. Assume that we have computed the lower bound under the assumption that J_i is scheduled in position k. To compute the lower bound under the assumption that J_i is scheduled in position $k-1$, we just have to exchange the assignments of job i and job $[k-1]$. The modification of the lower bound is then $\max(0, C_{[k-1]} - d_i) - \max(0, C_{[k]} - d_i) + \max(0, C_{[k]} - d_{[k-1]}) - \max(0, C_{[k-1]} - d_{[k-1]})$. Hence, we can "try" all possible positions for J_i in linear time. All impossible positions can thus be computed in $O(n^2)$.

4.2 Look-Ahead

We use a kind of look-ahead technique to test whether a job J_i can be removed of the set of possible first: The job J_i is sequenced immediately after P and a lower bound of the new scheduling situation (lb_2 in the current implementation) is computed. If this lower bound is greater than $\bar{T}$ then J_i cannot be first and it is removed from PF. A symmetric rule is used for the PL set.

5 Experimental Results

All techniques presented in this paper have been incorporated into a Branch-and-Bound method implemented on top of ILOG SOLVER and ILOG SCHEDULER. All experimental results have been computed on a PC Dell Latitude 650 MHz running Windows 98.

The instances have been generated with the scheme of Chu [8]. Each instance is generated randomly from three uniform distributions of r_i, p_i and d_i. The distribution of p_i is always between 1 and 10. The distributions of r_i and d_i depend on 2 parameters: α and β. For each job J_i, r_i is generated from the distribution $[0, \alpha \sum p_i]$ and $d_i - (r_i + p_i)$ is generated from the distribution $[0, \beta \sum p_i]$. Four values for α and three values for β were combined to produce 12 instances sets, each containing 10 instances of n jobs with $n \in \{10, 20, 30, 40, 50, \ldots, 500\}$ jobs.

In Table 3, Chu's lower bound is compared with the new lower bound lb_2 (see Section 3.4). Most times, $lb_1 = lb_{Chu}$ so, lb_1 has been skipped from the tables to simplify the presentation. To have a fair comparison of the bounds, the Branch-and-Bound procedure has been run without dominance property nor Intelligent Backtracking on instances with 20 jobs that are known to be hard ($\alpha = 0.5, \beta = 0.5$) [8]. Each line corresponds to a single instance and we report the optimal tardiness (Opt.), the value of the lower bound computed at the root of search tree (lb_{Chu}, lb_2), the relative gap (Gap) between the optimum and the two lower bounds, the number of backtracks (Bck.) and

	Chu's Lower Bound				lb_2			
Opt.	lb_{Chu}	Gap	Bck.	CPU	lb_2	Gap	Bck.	CPU
34	28	0.21	501340	773.9	34	0.00	1	0.1
111	46	1.41	***	***	111	0.00	0	0.1
115	35	2.29	***	***	93	0.24	535	2.1
95	47	1.02	603171	1313.3	81	0.17	10300	59.1
32	19	0.68	24913	56.6	31	0.03	221	1.0
12	9	0.33	1	0.1	12	0.00	0	0.0
79	40	0.97	***	***	73	0.08	284	1.6
192	167	0.15	10118	28.2	180	0.07	462	2.4
84	54	0.56	52922	79.8	79	0.06	6	0.1
24	16	0.50	69754	89.4	24	0.00	2	0.1

Table 3: Comparison Between lb_{Chu} and lb_2. $n = 20$ Jobs, $\alpha = 0.5$, $\beta = 0.5$, no Dominance Property, no Intelligent Backtracking.

the amount of CPU time in seconds required to solve the instance. A "***" in the table indicates that the search was interrupted after 1800 seconds. On the average, lb_{Chu} is at 81% of the optimum value while our lower bound is much closer (6%). The average number of backtracks over the instances solved by both methods has been divided by almost 12 and the CPU time by 37.

In Table 4, we show the efficiency of dominance properties, propagation rules and intelligent backtracking technique presented in this paper. For that, the Branch-and-Bound procedure has been run with lb_2 on instances with 30 jobs with various combinations of α and β. On each line of Table 4, the average results obtained over the 10 generated instances are reported. In columns 3 and 4 ("Chu"), we report the results obtained when the dominance properties of Chu are used [7, 8]. We then add (columns 5 and 6) the dominance and propagation rules presented in Section 2.1.4 (EQual Processing, EQP), Section 2.2 (Remove Dominated Sequences, RDS), Section 2.2.1 (optimiza-

		Chu		Chu, EQP, RDS 6-last, PIP		Chu, EQP, RDS 6-last, PIP, IB		Chu, EQP, RDS 6-last, PIP, IB LAH	
α	β	Bck.	CPU	Bck.	CPU	Bck.	CPU	Bck.	CPU
0	0.05	0	0.1	0	0.0	0	0.0	0	0.0
0	0.25	7	0.1	7	0.1	7	0.1	7	0.3
0	0.5	80	0.6	80	0.6	80	0.6	80	1.1
0.5	0.05	91	0.7	26	0.3	24	0.3	15	0.8
0.5	0.25	5420	32.8	215	3.2	156	2.5	62	7.2
0.5	0.5	11505	70.1	424	6.2	296	4.6	145	10.4
1	0.05	239	1.4	31	0.3	24	0.3	16	0.8
1	0.25	1724	7.1	46	0.5	39	0.5	25	1.3
1	0.5	2	0.0	2	0.0	2	0.0	2	0.1
1.5	0.05	24	0.2	7	0.1	6	0.1	5	0.3
1.5	0.25	560	2.3	11	0.2	8	0.2	7	0.6
1.5	0.5	2	0.1	1	0.1	1	0.0	1	0.1

Table 4: Efficiency of Dominance Properties, Propagation Rules and Intelligent Backtracking.

tion on the 6 last jobs) and Section 4.1 (Possible and Impossible Positions, PIB). Intelligent Backtracking (IB) is then added and results are reported in columns 7 and 8. Finally, the results obtained with the look-ahead (AH) technique are described in columns 9 and 10.

All "ingredients" described in this paper are useful to reduce the search space. However, the look ahead technique presented in Section 4.2 seems to be very costly in terms of CPU time compared to the corresponding reduction of the search tree. This is due to the relatively high complexity of the lower bound lb_2 that is used several times in the look ahead. We tried to use some weaker lower bound like lb_1 but it does not reduce the search space.

The results obtained with the version of the algorithm that incorporates all ingredients except the look-ahead technique are presented in Table 5. For each combination of parameters and for each value of n, we provide the average number of fails and the average computation time in seconds. A time limit of 3600 seconds has been fixed. All instances are solved within the time limit for up to 50 jobs. For $n = 60$, and for $(\alpha = 0.5, \beta = 0.5)$, most of the instances cannot be solved. As noticed earlier by Chu [8], instances generated according to this particular combination seem to be "hard" to solve in practice.

From this table, we can remark that the "hardness" increases very quickly with n, especially for ($\alpha = 0.5$, $\beta = 0.5$). For Each combination of parameters, we report the largest size of instance (column "Largest") for which 80% of instances are solved within one hour of CPU time. In practice most of the instances are solved within 30 seconds and our results compare well to those of [8]. For instance, the average number of fails for the combination $(\alpha = 0.5, \beta = 0.5)$ was greater than 36000 in [8], whereas this number is now lower than 300.

		$n = 10$		$n = 20$		$n = 30$		$n = 40$		$n = 50$		$n = 60$		Largest
α	β	Bck.	CPU	Bck.	CPU	Bck.	CPU	Bck.	CPU	Bck.	CPU	Bck.	CPU	
0	0.05	0.0	0.01	0	0.02	0	0.0	1	0.1	1	0.2	0	0.1	300
0	0.25	0.3	0.01	3	0.03	7	0.1	22	0.5	17	1.2	55	3.2	120
0	0.5	1.1	0.01	7	0.07	80	0.6	57	1.7	304	7.7	1591	91.3	80
0.5	0.05	2.9	0.02	16	0.08	24	0.3	73	1.8	96	3.9	238	19.7	90
0.5	0.25	3.6	0.02	26	0.18	156	2.5	484	21.4	1530	175.3	5253	1083.0	60
0.5	0.5	3.7	0.01	43	0.30	296	4.6	2366	131.7	9438	931.0	***	***	50
1	0.05	1.3	0.02	13	0.07	24	0.3	52	1.4	93	4.5	128	25.7	90
1	0.25	2.0	0.02	14	0.08	39	0.5	86	2.2	90	4.4	237	28.1	500
1	0.5	0.6	0.01	22	0.15	2	0.0	11	0.2	14	0.3	5	0.5	500
1.5	0.05	1.3	0.02	5	0.02	6	0.1	30	1.1	37	2.2	40	6.7	190
1.5	0.25	0.1	0.02	2	0.02	8	0.2	2	0.1	37	0.8	6	1.6	500
1.5	0.5	0.0	0.02	0	0.01	1	0.0	0	0.1	0	0.2	0	0.5	500

Table 5: Results Obtained with up to 60 Jobs.

6 Conclusion

We have presented new lower-bounds, new constraint propagation techniques and new dominance properties for $1|r_i|\sum T_i$. Computational results show that the proposed approach outperforms the best known procedures. We think that several techniques presented in this paper can be extended to more complex criteria such as weighted tardiness or weighted flow time.

Acknowledgments

The authors would like to thank Chengbin Chu, Federico Della Croce, Filippo Focacci and Andrea Grosso for enlightening discussions on total tardiness.

References

[1] T.S. Abdul-Razacq, C.N. Potts and L.N. Van Wassenhove, *A survey of algorithms for the single machine total weighted tardiness scheduling problem*, Discrete Applied Mathematics, 26, pp 235-253 (1990).

[2] Ph. Baptiste, C. Le Pape and W. Nuijten, *Constraint-Based Scheduling, Applying Constraint Programming to Scheduling Problems*, International Series In Operations Research And Management Science, Volume 39, Kluwer (2001).

[3] Ph. Baptiste and A. Jouglet, *On minimizing total tardiness in a serial batching problem*, RAIRO Operational Research, 35, pp 107-115 (2001).

[4] J. Carlier, *Ordonnancements à contraintes disjonctives*, RAIRO, 12, pp 333-351 (1978).

[5] J. Carlier and E. Pinson. *A Practical Use of Jackson's Preemptive Schedule for Solving the Job-Shop Problem*, Annals of Operations Research, 26, pp 269-287 (1990).

[6] S. Chang, Q. Lu, G. Tang and W. Yu, *On decomposition of the total tardiness problem*, Operations Research, 17, pp 221-229. (1995).

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

[8] C. Chu, *A Branch-and-Bound algorithm to minimize total tardiness with different release dates*, Naval Research Logistics, 39, pp 265-283,(1992).

[9] J. Du, J.Y.T. Leung, *Minimizing total tardiness on one processor is NP-Hard*, Mathematics of Operations Research, 15, pp 483-495 (1990).

[10] H. Emmons, *One-machine sequencing to minimize certain functions of job tardiness*, Operations Research, 17, pp 701-715 (1969).

[11] F. Focacci, *Solving Combinatorial Optimization Problems in Constraint Programming*, University of Ferrara, PhD Thesis, pp 101-104 (2000).

[12] E.L. Lawler, *A pseudo-polynomial algorithm for sequencing jobs to minimize total tardiness*, Annals of Discrete Mathematics, 1, pp 331-342 (1977).

[13] O. Lhomme. *Consistency Techniques for Numeric CSPs*, Proc. 13th International Joint Conference on Artificial Intelligence, (1993).

[14] C.N. Potts, L.N. Van Wassenhove, *A decomposition algorithm for the single machine total tardiness problem*, Operations Research Letters, 26, pp. 177-182, (1982).

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

[16] W. Szwarc, F. Della Croce and A. Grosso, *Solution of the single machine total tardiness problem*, Journal of Scheduling, 2, pp 55-71 (1999).