An efficient algorithm for the earliness-tardiness scheduling problem

Francis Sourd * Safia Kedad-Sidhoum †

September 7, 2005

Abstract

This paper addresses the one-machine scheduling problem with earliness-tardiness penalties. We propose a new branch-and-bound algorithm that can solve instances with up to 50 jobs and that can solve problems with even more general non-convex cost functions. The algorithm is based on the combination of a Lagrangean relaxation of resource constraints and new dominance rules.

1 Introduction

A useful but difficult criterion in scheduling theory is the minimization of both earliness and tardiness costs. It comes from the "just-in-time" philosophy in management and production theory: an item should be delivered when it is required by the customer. Therefore both early and tardy delivery of a task with respect to its due date is penalized. Recent surveys on these problems have been published by Hoogeveen [17, section 4], Lauf and Werner [24] and Gordon, Proth and Chu [14].

In this paper, we consider the basic single-machine earliness-tardiness problem, which is denoted by $1\,|\,r_j\,|\,\sum \alpha_j E_j + \beta_j T_j$ in the standard three-field notation. A set $\mathcal{J}$ of n jobs $J_1, \ldots, J_n$ with positive processing times $p_1, \ldots, p_n$ has to be scheduled with no preemption on a single machine. For any j, J_j is available at time r_j and has a due date d_j. If the completion time of J_j, denoted by C_j, occurs after d_j, the job is late and a penalty $\beta_j(C_j - d_j)$ must be paid. Conversely, if $C_j < d_j$, the job J_j is early and the penalty is then $\alpha_j(d_j - C_j)$. The goal is to find a schedule that minimizes the sum of all the penalties. Although we focus on this earliness-tardiness problem, the algorithm presented in this paper is still valid in the more general case where the cost for scheduling J_j is given by some function $f_j(C_j)$ which may be non-convex. In particular, our algorithm can solve some extensions of the models presented in the literature: time windows [11], due windows [38], non-execution penalties [15], prohibition of idle time [25, 36]. In the earliness-tardiness case, we simply have $f_j(C_j) = \max(\alpha_j(d_j - C_j), \beta_j(C_j - d_j))$, which is a convex piecewise linear function with two segments.

*Francis.Sourd@lip6.fr – LIP6, CNRS, 8 rue du Capitaine Scott, 75015 Paris, France.

†Safia.Kedad-Sidhoum@lip6.fr – LIP6, Université Pierre et Marie Curie, 8 rue du Capitaine Scott, 75015 Paris, France

Since $1||\sum T_j$ is NP-hard [10], this problem is clearly NP-hard and, as observed in the many surveys [17, 24, 14, 3], polynomial cases are very rare. Most of them assume that there is a common due date, that is $d_j = d$ for all jobs. The most general polynomial-time algorithm for a problem with distinct due dates is due to Verma and Dessouky [37] and it can solve the case where $p_j = p$, $\alpha_1 \leq \cdots \leq \alpha_n$ and $\beta_1 \leq \cdots \leq \beta_n$. Hassin and Shani [15] present a list of existing polynomial time algorithms for earliness-tardiness problems as well as some extensions.

We are interested in an exact approach for the general case. The first branch-and-bound procedure was proposed by Fry et al. [13] for the case $\alpha_j = \alpha$ and $\beta_j = \beta$ and was able to solve instances with up to 12 jobs. The special case $\alpha_j = \beta_j = 1$ is studied by Kim and Yano [22] and Fry et al. [12] who proposed algorithms viable for 20 and 25 jobs respectively. Hoogeveen and van de Velde [18] proposed six lower bounds and several dominance rules for the case $\alpha_j = \alpha$ and $\beta_j = \beta$ but cannot solve instances with more than 25 jobs. For the same problem, Chen and Lin [7] presented a long list of dominance rules and experiments for instances with 20 jobs. Finally, another algorithm was proposed by Chang [5] for the symmetric case ($\alpha_j = \beta_j$) that could solve all the instances up to 35 jobs and some instances with 40 and 45 jobs — however, the parameters used in the generation of the instances tend to indicate that these instances are rather easy.

For the general case, Sourd and Kedad-Sidhoum [33] proposed a branch-and-bound procedure based on a pseudo-polynomial time lower bound based on an assignment problem. A similar lower bound was reported by Bulbul et al. [4]. This lower bound can be computed in $O(n^2T)$ time where T is the horizon of the schedule. This is significantly more time consuming than the previously proposed lower bounds but the lower bound is also significantly better. The resulting branch-and-bound algorithm is able to solve the instances with up to 30 jobs. Sourd [30] showed that a polynomial-time lower bound can be derived from this approach but, unfortunately, it does not help to solve instances with more jobs. A last algorithm, reported by Tanaka et al. [35], also fails to solve instances with 30 jobs or more. Finally, we can mention the approach of Liaw [25] —recently generalized by Valente and Alves [36]— to solve the problem with no idle time inserted but, even in this easier case, instances with more than 30 jobs seem to be out of reach.

The contribution of this article is to show that larger instances can be solved. Indeed, in our experimental tests, almost all the 40-job instances and 85% of the 50 jobs instances are solved within 1000 seconds. This result is due to the usage of an improved lower bound, which is even longer to compute than the above-mentioned ones [33, 30] but which is better. This lower bound relies on a Lagrangean relaxation of the *time-indexed formulation* of Sousa and Wolsey [34]. Two relaxed problems can be derived from this formulation. First, the number of occurrences of the jobs in the schedule can be relaxed. This approach was followed by Peridy et al. [28] to minimize the number of late jobs and by Avella et al. [2] to minimize the weighted completion time. The second approach, introduced by Christofides et al. [8], consists of relaxing the resource constraints. Much work was devoted to the resolution of the Lagrangean problem, the convergence of the subgradient and its usage to find good or optimal solutions [26, 6, 20, 27]. However, to the best of our knowledge, all these results are related to multiple machine environments so that it was a surprise for us that this type of lower bound is also efficient in the simple single machine case without precedence constraints.

Lagrangean relaxations are usually time-consuming so that it is often challenging to use them

		$n = 20$	$n = 30$	$n = 40$	$n = 50$	All
HEUR1	Optimum found	62.1 %	38.4 %	23.5 %	17.4 %	36.0 %
	Mean deviation	1.06 %	1.42 %	1.46 %	1.59 %	1.38 %
	Max deviation	21.9 %	20.5 %	22.3 %	16.2 %	22.3 %
HEUR10	Optimum found	89.4 %	75.8 %	62.6 %	55.7 %	71.4 %
	Mean deviation	0.12 %	0.20 %	0.26 %	0.28 %	0.21 %
	Max deviation	8.76 %	10.4 %	6.47 %	7.63 %	10.4 %
HEURn	Optimum found	91.9 %	86.3 %	80.6 %	76.4 %	84.1 %
	Mean deviation	0.08 %	0.09 %	0.10 %	0.11 %	0.10 %
	Max deviation	8.76 %	10.0 %	5.53 %	4.10 %	10.0 %

Table 1: Efficiency evaluation of the descent algorithms

in a branch-and-bound algorithm. The main achievement of our work is to efficiently integrate this lower bound in the search algorithm and to propose new dominance rules. Section 2 describes all the components of the branch-and-bound algorithms (initial upper bound, lower bound, dominance rules and branching scheme). Experimental results and comparison to other approaches are presented in Section 3.

2 The branch-and-bound algorithm

2.1 Initial solution

The preliminary step of the algorithm is a *heuristic search* of a good solution in order to have a good upper bound for the optimum. Through previous experience [31], we came to the conclusion that a simple iterated descent algorithm with a generalized pairwise interchange neighborhood is decently efficient. By generalized pairwise interchange, we mean that the solutions are encoded by the job sequence and these sequences are modified by either a pairwise job swap or an "extract and relocate" operation. The cost of the schedule when the sequence is modified can be efficiently computed using the ideas of Hendel and Sourd [16].

As this simple heuristic was never compared to optimal solutions, Table 1 shows the performance of three variants of the algorithm tested on a set of 4914 instances whose optimum are known (see Section 3.1 for details about instance generation). HEUR1 is a simple descent algorithm from a random sequence. HEUR10 consists of iterating HEUR1 ten times while HEURn iterates HEUR1 n times, where n is the number of jobs of the instance. For each algorithm, the three lines respectively indicate the number of times the heuristic found the optimum, the mean deviation and the max deviation from the optimum according to the size of the instances.

The experiments confirm the high efficiency of this very basic heuristic search algorithm. Indeed, the performances of HEUR1 are decent. However, while HEUR1 may return a bad local optimum (as indicated by the maximal deviation), HEUR10 and *a fortiori* HEURn very often find the optimum and the mean deviation is very weak.

2.2 Lower bound

We first present the time indexed formulation of our problem. As mentioned in the introduction, the lower bound is derived from a Lagrangean relaxation of this mixed integer program (MIP). We first introduce the horizon T which must be an upper bound on the makespan of an optimal schedule of the problem. In earliness-tardiness scheduling, we can choose $T = \max_j d_j + \sum_j p_j$. The notation $[t, t']$ is used to designate the set of the integers between t and t'. For any $t \in [r_j, T - p_j]$, let x_{jt} be a binary variable equal to 1 if the task J_j starts at time t and 0 otherwise. Let us define $est_j(t) = \max(r_j, t - p_j + 1)$. $est_j(t)$ is clearly the earliest start time of J_j such that it may be processing in time slot t.

$$\begin{aligned}
\min \quad & \textstyle\sum_{j\in\mathcal{J}} \sum_{t=r_j}^{T-p_j} f_j(t-p_j)x_{jt} && && (1)\\
\text{s.t.} \quad & \textstyle\sum_{t=r_j}^{T-p_j} x_{jt} = 1 && \forall\, j \in \mathcal{J} && (2)\\
& \textstyle\sum_{j\in\mathcal{J}} \sum_{s=est_j(t)}^{t} x_{js} \le 1 && \forall\, t \in [0, T-1] && (3)\\
& x_{jt} \in \{0,1\} && \forall\, j \in \mathcal{J}\,,\ \forall\, t \in [r_j, T-p_j] && (4)
\end{aligned}$$

Equations (2) ensure that each job is executed once. Inequalities (3), also refered to as *resource constraints*, state that at most one job can be handled at any time. Clearly, this formulation allows the occurrence of idle time. The objective (1) renders the cost of the schedule.

The lower bound is derived from the Lagrangean relaxation of the resource constraints (3). Our choice was motivated by the the study of Kedad-Sidhoum et al. [21] in the parallel machine case which shows that this Lagrangean relaxation gives better results than the relaxation of constraints (2). Lagrangean multipliers $\mu = (\mu_0, \cdots, \mu_{T-1}) \ge 0$ are introduced for each t so that, for each vector $\mu \ge 0$, a lower bound denoted by $\mathcal{L}(\mu)$ is obtained by solving the so-called Lagrangean problem:

$$\begin{aligned}
\min_{x_{jt}} \quad & \sum_{j\in\mathcal{J}} \left(\sum_{t=r_j}^{T-p_j} f_j(t-p_j)x_{jt} + \mu_t \left(\sum_{s=est_j(t)}^{t} x_{js} - 1 \right) \right) && (5)\\
& \text{subject to the constraints (2) and (4)}
\end{aligned}$$

Once the resource constraints are relaxed, the resulting Lagrangean problem consists of scheduling "unconstrained" tasks whose starting costs are defined by $c_{jt} = f_j(t-p_j) + \sum_{s=t}^{t+p_j-1} \mu_t$. Clearly J_j must start at time t where c_{jt} is minimum. This start time can be computed in $O(T)$ time by observing that $c_{j,t+1} = c_{jt} + f_j(t+1-p_j) - f_j(t-p_j) - \mu_t + \mu_{t+p_j}$. Consequently, $\mathcal{L}(\mu)$ is computed in $O(nT)$.

Good multipliers have then to be computed in order to maximize $\mathcal{L}(\mu)$. This phase is usually performed through a subgradient method. Obviously, the convergence speed of this phase is of key importance because the computation of the lower bound is processed at each node of the branch-and-bound procedure. We now present a heuristic to find a priori good multipliers.

The heuristic is based on the initial solution, denoted by σ, computed by the heuristic of Section 2.1. Roughly speaking, the multipliers are approximated by a piecewise linear function whose slopes depend on the job in process in σ. More precisely, they are initialized by Algorithm 1.

Algorithm 1: Heuristic to produce good Lagrangean multipliers

Input: a good feasible schedule σ
Output: good Lagrangean multipliers $\mu_0, \ldots, \mu_{T-1}$
$\mu_T = 0$
for $t = T-1$ **downto** *0* **do**
  if *a job is in process at* t *in* σ **then**
    let J_j be this job in process
    if $t \geq d_j$ **then** $\mu_t = \mu_{t+1} + \beta_j/p_j$ **else** $\mu_t = \mu_{t+1} - \alpha_j/p_j$
  end
  $\mu_t = 0$
end

In order to give a theoretical justification of this heuristic algorithm, we provide the following proposition.

Proposition 1. *In the case where $d_j = 0$ and $r_j = 0$ for all jobs (that is we have an instance of $1||\sum w_j C_j$), the heuristic finds the optimal multipliers when it is run with Smith's schedule as input schedule σ.*

Proof. As $d_j = 0$, earliness penalties are of course irrelevant here. For the simplicity of the proof, we assume that $\beta_1/p_1 > \beta_2/p_2 > \cdots > \beta_n/p_n > 0$ but the result is still valid when two jobs have the same ratio β_j/p_j. In the optimal schedule σ provided by Smith's rule, the completion time of J_j is $C_j(\sigma) = \sum_{i=1}^{j} p_i$. Let μ be the vector of Lagrangean multipliers initialized by the heuristic. We show that the optimal solution of the Lagrangean problem $\mathcal{L}(\mu)$ is to complete each job at $C_j(\sigma)$ for each j. To this end, let us consider a solution for $\mathcal{L}(\mu)$ and let us assume that $x_{jt} = 1$ for some $t < C_j(\sigma) - p_j$. By modifying the solution such that $x_{j,t+1} = 1$ (and thus $x_{jt} = 0$), the cost increases by $c_{j,t+1} - c_{jt} = \beta_j + \mu_{t+p_j} - \mu_t = \beta_j + \sum_{\theta=t}^{t+p_j-1}(\mu_{\theta+1} - \mu_\theta)$. Since $\theta + 1 \leq t + p_j < C_j$, the job J_i in process at θ is such that $\beta_i/p_i \geq \beta_j/p_j$ so that $\mu_{\theta+1} - \mu_\theta = -\beta_i/p_i \leq -\beta_j/p_j$. Therefore $\sum_{\theta=t}^{t+p_j-1}(\mu_{\theta+1} - \mu_\theta) \leq -\beta_j$, which proves that the cost increase is negative. Similarly, if $x_{jt} = 1$ for some $t > C_j - p_j$, changing the solution by $x_{j,t-1} = 1$ also results in a cost decrease. Therefore, an optimal solution is to set $x_{jt} = 1$ if and only if $t = C_j(\sigma)$. As this solution satisfies the relaxed resource constraints, it is both primal and dual optimal. □

In practice, the heuristic finds good multipliers as illustrated by Figure 1. The graph compares the multipliers found by the heuristic (the piecewise linear function) with the multiplier values obtained after the convergence of the subgradient method (the highly irregular curve). With such an initialization of the multipliers, a very good lower bound is obtained after a small number of iterations of the subgradient method (typically, the convergence loop is stopped after n iterations).

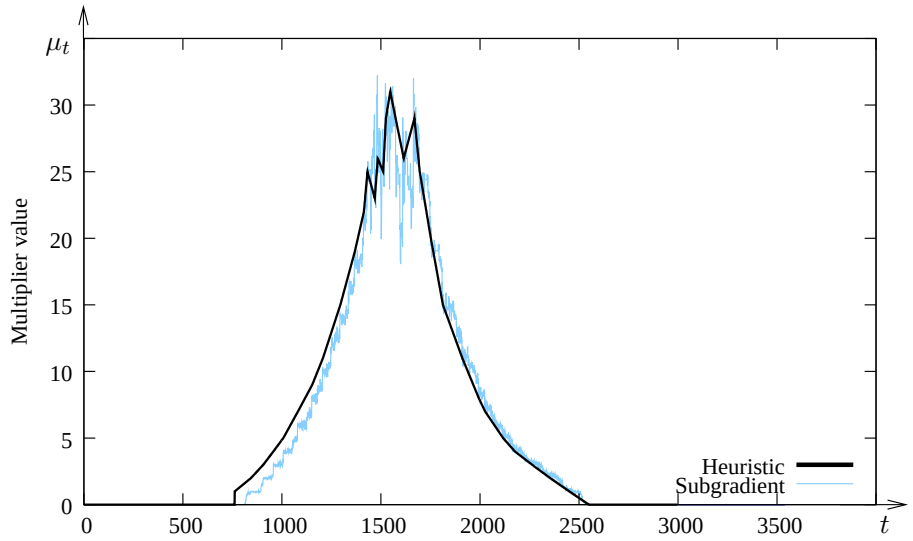

Figure 1: Heuristic and optimized Lagrangean multipliers

2.3 Branching scheme

The *branching scheme* builds the sequence of tasks from the first to the last one. In other words, at a node of depth k in the search tree, we have an initial subsequence σ of k tasks while the other $n-k$ tasks must be scheduled after all the sequenced tasks. The cost of the initial subsequence is modeled by a function Σ_σ which is called the *initial cost function*. More precisely, $\Sigma_\sigma(t)$ is the minimal cost such that all the tasks in the subsequence σ are completed before t. The definition domain of Σ_σ is $[C_{\max}(\sigma), T)$ where $C_{\max}(\sigma)$ is the earliest completion time (*makespan*) of the initial subsequence and T is the horizon. In order to have lighter notations, we assume that $\Sigma_\sigma(t) = \infty$ for any t that is out of the definition domain.

When the cost function f_j is general, the values of Σ_σ must be computed for any integer time point in $[C_{\max}(\sigma), T-1]$. In the earliness-tardiness case, Σ_σ is a convex, non-increasing, piecewise linear function with at most k segments [32], which can be more efficiently encoded in $O(k)$ space. More generally, Σ_σ is piecewise linear when the cost functions f_j are piecewise linear and Σ_σ is convex when the cost functions f_j are convex.

When a child node is created, a job —say J_j, which is not in σ— is added at the end of the sequence σ. This operation creates a new subsequence denoted by $\sigma \oplus J_j$. The new initial cost function $\Sigma_{\sigma\oplus J_j}$ is derived from Σ_σ by a very simple dynamic programming procedure. This operation takes $O(T)$ time in the general case, but only $O(k)$ time in the earliness-tardiness case [32].

At any node of the search tree, the computation of the lower bound must be adapted in order to take into account the presence of the initial subsequence σ. For a given set of multipliers λ, the lower bound is given by $\min_\theta (\Sigma_\sigma(\theta) + \mathcal{L}_\theta(\lambda))$ where $\mathcal{L}_\theta(\lambda)$ is the cost of the Lagrangean relaxation for scheduling the non-sequenced tasks in the interval $[\theta, T)$, that is we add the constraint $x_{jt} = 0$ for all non-sequenced jobs J_j and all $r_j \leq t < \theta$ (and we remove the variables related to sequenced jobs).

In order to compute $\mathcal{L}_t(\lambda)$, we compute, for all θ and all non-sequenced jobs J_j, the values $c'_{j\theta} = \min_{\theta \geq t} c_{j\theta}$ which is done in $O((n-k)T)$ time by using the relationship $c_{j,t+1} = c_{jt} - \mu_t + \mu_{t+p_j}$. Then, all the values $\mathcal{L}_t(\lambda)$ for $0 \leq t < T$ can be computed in $O((n-k)T)$ and therefore the lower bound is not longer to compute than the root lower bound.

Moreover, very good multipliers for a node can be heuristically derived by taking the multipliers of the parent node.

2.4 Dominance rules

2.4.1 Dominating initial cost function

Our *dominance rule* consists of checking whether the initial subsequence σ is dominated by another subsequence or by the conjunction of several subsequences. Moreover, we also propagate some information about partially dominated schedules in order to restrict the set of possible completion times of some jobs and, by this way, to reinforce the lower bound.

Let $J(\sigma)$ be the set of jobs contained in σ. If the three following conditions are satisfied

- σ' is a subsequence such that $J(\sigma') = J(\sigma)$,
- $\Sigma_{\sigma'}(t) \leq \Sigma_\sigma(t)$ for all $t \geq 0$ and
- $\Sigma_{\sigma'}(t) < \Sigma_\sigma(t)$ for some $t \geq 0$

then σ is dominated by σ'. Indeed, for any solution starting by the subsequence σ, we can get a better or equal solution by reordering the initial subsequence according to σ'. Therefore, the node associated to the subsequence σ can be discarded.

More generally, let S be a set of subsequences such that $\sigma \notin S$ and, for any $\sigma' \in S$, $J(\sigma') = J(\sigma)$. Let $\Sigma_S(t) = \min_{\sigma' \in S} (\Sigma_{\sigma'}(t))$. The dominance is formally proved in the next proposition.

Proposition 2. *Let S be a set of subsequences such that $\sigma \notin S$. If the three conditions are satisfied*

- *$J(\sigma') = J(\sigma)$ for any $\sigma' \in S$,*
- *$\Sigma_S(t) \leq \Sigma_\sigma(t)$ for all $t \geq 0$,*
- *$\Sigma_S(t) < \Sigma_\sigma(t)$ for some $t \geq 0$,*

then σ is dominated by S, which means that for any solution starting by the subsequence σ, a better —or equal— solution can be derived by reordering the initial tasks according to some subsequence in S.

Proof. Let us consider a feasible solution starting by the subsequence σ. Without loss of generality, we can assume that the jobs are sequenced in the order $J_1, J_2, \ldots, J_n$. Let k be the number of jobs in the subsequence σ. Clearly, $\sigma(k) = J_k$. By construction, there exists a subsequence σ' in S such that $\Sigma_{\sigma'}(C_k) \leq \Sigma_\sigma(C_k)$. Therefore, we can schedule the jobs $J_1, \ldots, J_k$ before C_k and let the completion times of the other jobs unchanged. Clearly, the cost of this new schedule is not greater than the initial one. □

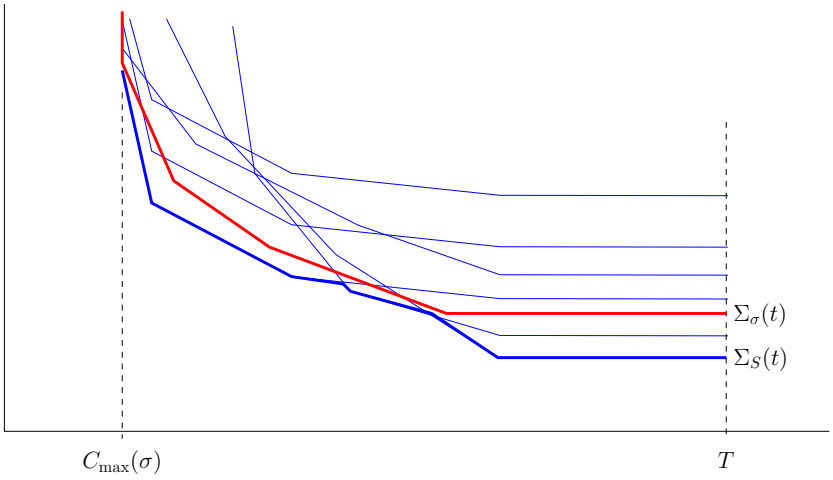

Figure 2: A dominated subsequence

In our implementation, at any node, a set S of subsequences is heuristically built, it corresponds to a neighborhood of σ in which the last job of the subsequence is either re-inserted in an earlier position or swapped with an earlier job. For instance, with $\sigma = (J_1, J_2 \ldots, J_k)$ then we use

$$S(\sigma) = \{(J_k, J_1, J_2, \ldots, J_{k-1}), (J_1, J_k, J_2, \ldots, J_{k-1}), \ldots, (J_1, J_2, \ldots, J_k, J_{k-1})\}$$
$$\cup \{(J_k, J_2, \ldots, J_{k-1}, J_1), (J_1, J_k, \ldots, J_{k-1}, J_2), \ldots, (J_1, \ldots, J_k, J_{k-1}, J_{k-2})\}$$

As illustrated by Figure 2, the so called *dominating initial cost function* $\Sigma_S(t)$ is a non-convex piecewise linear function. To prevent bad behaviors due to numerical instability, the dominance rule is activated only if $\Sigma_S(t) + \epsilon < \Sigma_\sigma(t)$ for some small positive ϵ.

In order to improve the dominating initial cost function and to avoid redundant calculations, the pair $(J(\sigma), \min(\Sigma_S, \Sigma_\sigma))$ can be stored —in a hash table in our implementation— so that the information about dominating schedules can be used at subsequent nodes. Note that this information is stored even if the current initial sequence is not dominated. Thus, when the branch-and-bound algorithm opens a node with an initial sequence $\sigma^\star$, it first searches in the hash table for a pair of the form $(J(\sigma^\star), \Sigma)$. If such a function Σ exists, we first check whether $\Sigma < \Sigma_\sigma$. If successful, this quick test proves that σ is dominated. Otherwise, we can take $\min(\Sigma, \Sigma_{S(\sigma^\star)})$ as dominating initial cost function, which is clearly better than $\Sigma_{S(\sigma^\star)}$.

It must be observed that the storage of all these functions requires exponential space. For medium-size instances, computation times are significantly improved (by about 20%) but, for more difficult instances (typically instances with 50 jobs or more), memory limitation prevents this approach to work well unless an elaborated policy for storage management is implemented.

2.4.2 Improving the initial cost function

We now show how the lower bound, and more precisely the initial cost function Σ_σ, can be improved when the current subsequence σ is not dominated by the dominating initial cost

function Σ_S. The key idea is that, even if we do not have $\Sigma_S(t) < \Sigma_\sigma(t)$ for all t, there may be time intervals over which the inequality is satisfied. In such a case, the subsequence is said to be *partially* dominated. Without loss of generality, we assume again that $\sigma = (J_1, \ldots, J_k)$.

Let us first assume that for some $\theta > 0$, we have that $\Sigma_S(t) < \Sigma_\sigma(t)$ for all $t \in [0, \theta)$. Thus, any schedule starting with σ such that $C_k < \theta$ is dominated. In the current node subproblem, we are not interesting in finding solutions such that $C_k < \theta$ so that we can restrict the definition domain of Σ_σ to be equal to $[\theta, T)$ or even to $[\lceil\theta\rceil, T)$ if the r_i, d_i and p_i are integer. Clearly, this change in the current subproblem does not discard any dominating optimal solution and it may improve the quality of the lower bound since Σ_σ is used in the calculation of the lower bound.

Similarly, we consider the case where, for some $\theta > 0$, we have that $\Sigma_S(t) < \Sigma_\sigma(t)$ for all $t \geq \theta$. In the current node, we are then interested only in the schedules such that $C_k \leq \theta$. We can then change the right end of the definition domain of Σ_σ to θ (or $\lfloor\theta\rfloor$).

More generally, we can remove from the definition domain of Σ_σ any interval on which Σ_σ is dominated by Σ_S. The computation of the lower bound in presence of Σ_σ, which is described in Section 2.3, can be trivially adapted in order to deal with these holes in the definition domain.

3 Experimental results

The algorithm was implemented in C++ (it can be compiled with MSVC7 or gcc 3.3). It is available on the internet[1]. In the tested version, the initial heuristic is HEUR10 and the dominating initial cost functions are not stored for reasons given in Section 2.4.1.

3.1 Instances

The generation scheme of the test instances is based on those used in the literature [18, 33]. In order to avoid to have too many parameters, all the release dates are null. The difficulty of instances with release dates is tackled at the end of this section (in section 3.3). For a given value of n, the processing times of each job are first randomly drawn from the uniform distribution $U[10, 100)$. The due date are drawn from $U[d_{\min}, d_{\min} + \rho P]$ where $d_{\min} = \max(0, P(\tau - \rho/2))$ and $P = \sum_{j=1}^{n} p_j$. The two parameters τ and ρ are respectively the *tardiness* and *range* parameters.

We generated instances for $n \in \{20, 30, 40, 50\}$, $\tau \in \{0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$ and $\rho \in \{0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$. For each value of (n, τ, ρ), 26 instances were generated, which induces a total of 5096 instances.

3.2 Results

As for any branch-and-bound algorithm, the computation times required to solve two different instances, even if generated with the same parameters, may vary a lot. Therefore, the average computation time is not very relevant and we prefer to present our results with a runtime distribution diagram: for a given CPU time t on the log-scale abscissa axis, we read in ordinate the

[1] `http://www-poleia.lip6.fr/~sourd/project/et`

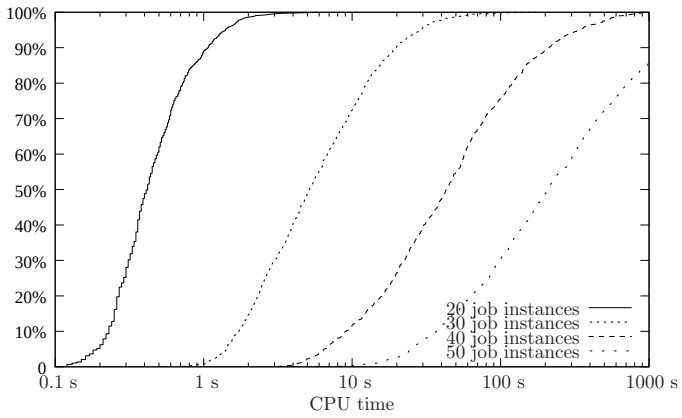

Figure 3: Percentage of solved instances

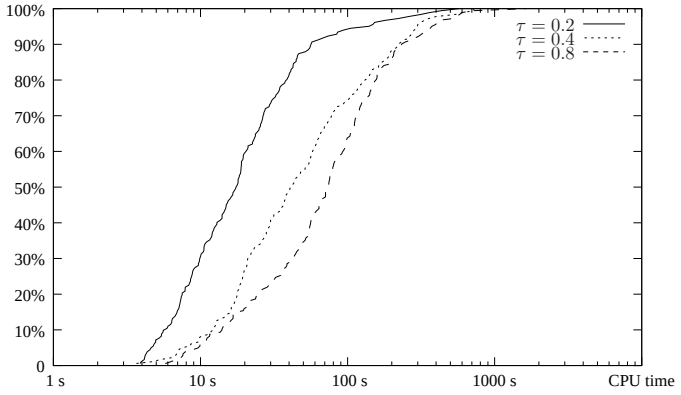

Figure 4: Runtime distribution of 40-job instances when τ varies

percentage of instances that were solved within the time t. Therefore the run time distribution is a non decreasing graph which starts from the origin and tends to 100% when t is infinite.

Figure 3 shows the runtime distributions to respectively solve the instances with 20, 30, 40 and 50 jobs. Clearly, this new algorithm outperforms previous exact algorithms whose limits are around to 30-job instances: indeed, all the 40-job instances and more than 85% of the 50 job instances are solved within 1000s. The algorithms of Sourd and Kedad-Sidhoum [33, 30] are able to efficiently solve instances with 30 jobs but the number of nodes is significantly more important. This observation explains why these algorithms fail to solve most instances with 40 jobs while our new algorithm succeeds.

Figure 4 compares the runtimes to solve the instances with 40 jobs for different values of the tardiness factor τ. It indicates that the instances are generally harder when the tardiness factor is greater, which means that the due dates are distant. It can be explained by observing that the jobs scheduled after the last due date are ordered according to the ratio of p_i/β_i. Therefore, when the last due date is earlier, the instance should be easier. Furthermore, we also observe that the optimal solution of the instances with $\tau = 0.8$ have some idle time inserted before the first task so we can conclude that instances with $\tau > 0.8$ are not more difficult. It roughly

	$\rho = 0.2$	$\rho = 0.4$	$\rho = 0.6$	$\rho = 0.8$	Total
BnB	10/20	13/20	14/20	14/20	51/80
MIP	19/20	5/20	0/20	0/20	24/80

Table 2: Number of solved instances within 100s when $\alpha_j = \alpha$ and $\beta_j = \beta$

corresponds to the case of a *large* or *unrestricted* due date in common due date scheduling.

The same diagram to compare the runtimes for different values of the range factor ρ does not lead to any valuable conclusion: the curves are very close and interlaced.

3.3 Special cases

We first consider the case where the penalties are job independent ($\alpha_j = \alpha$ and $\beta_j = \beta$ for all j) [18, 7]. Indeed, this problem can be efficiently formulated as a MIP with *positional and assignment variables* [23, 29]. This formulation is known to have a quite good linear relaxation while it has a small (polynomial) number of variables. To the best of our knowledge, this formulation has not been applied to irregular criteria. Therefore, it is interesting to test the MIP for this problem. The *assignment* variable y_{jk} is equal to one if job J_j is processed in position k, and zero otherwise. The *positional* completion time $C_{[k]}$ is the completion time of the job in position k. The positional earliness and tardiness penalties $E_{[k]}$ and $T_{[k]}$ are similarly defined.

$$\begin{aligned}
\min \quad & \textstyle\sum_k \alpha E_{[k]} + \beta T_{[k]} && && (6)\\
\text{s.t.} \quad & T_{[k]} - E_{[k]} = C_{[k]} - \textstyle\sum_j y_{jk} d_j && \forall k && (7)\\
& \textstyle\sum_j y_{jk} = 1 && \forall k && (8)\\
& \textstyle\sum_k y_{jk} = 1 && \forall j && (9)\\
& C_{[k]} \geq \textstyle\sum_j y_{jk}(r_j + p_j) && \forall k && (10)\\
& C_{[k]} \geq C_{[k-1]} + \textstyle\sum_j y_{jk} p_j && \forall k > 1 && (11)\\
& y_{jk} \in \{0, 1\} && \forall j, k && (12)\\
& E_{[k]} \geq 0, \quad T_{[k]} \geq 0 && \forall k && (13)
\end{aligned}$$

Equations (7) and (13) define the relationship between the positional completion times and the positional earliness and tardiness — observe that $\sum_j y_{jk} d_j$ is the due date of the job in position k. Equations (8) and (9) are the classical assignment constraints. Equations (10) force each job to start after its release dates and equations (11) require that a job cannot start before its predecessor is completed in the machine sequence.

We generated 40-job instances with α and β randomly chosen in $\{1, \ldots, 5\}$. For each pair $(\tau, \rho) \in \{0.2, 0.4, 0.6, 0.8\}^2$, five instances were generated which leads to a total of 80 instances. The MIP model was solved by ILOG CPLEX 9.1. Table 2 indicates, for each value of ρ, the number of instances solved within 100s by our algorithm (BnB) and by ILOG CPLEX (MIP). Clearly, the MIP is very efficient to solve instances where all the due dates are in a short time interval because all but one instances with $\rho = 0.2$ can be solved with the MIP formulation while

BnB only solves half the instances. Moreover, MIP is usually faster. However, for instances with higher range values, the MIP is no more efficient and the BnB should be prefered.

The algorithm was also tested on a class of difficult instances for the problem $1|r_j|\sum w_jT_j$ [9, 1]. The best dedicated algorithms for this problem can solve all the instances with 30 jobs or less [19]. Our approach is of course less efficient: some instances with 20 and 25 jobs cannot be solved. The main reason of this moderate performance could be that there are several dominances between early jobs that are not taken into account by our algorithm. However, even if earliness penalties are added, the presence of release dates makes these instances more difficult than those generated in Section 3.1.

4 Conclusion

This paper presents a new branch-and-bound algorithm for the one-machine earliness-tardiness problem. This algorithm outperforms previous ones because the lower bound is better so that less nodes are required in the search tree. Moreover, new dominance rules also help in reducing the size of the search tree.

Experimental results in Section 3.3 suggest that the algorithm could be improved to solve special cases where the range between due dates is narrow and were earliness penalties are significantly smaller than tardiness ones. The first case could be improved by a dedicated branching scheme while the second case could take advantage of new dominance rules.

In a theoretical point of view, it would be interesting to have a better understanding of the optimal values of the Lagrangean multipliers, in order to have, for example, a performance guarantee on the lower bound.

Acknowledgments

Thanks are due to Federico della Croce who suggested the authors to test the MIP formulation in Section 3.3.

References

[1] M.S. Akturk and D. Ozdemir, *An exact approach to minimizing totalweighted tardinesswith release dates*, IIE Transactions **32** (2000), 10911101.

[2] P. Avella, M. Boccia, and B. DAuria, *Near-optimal solutions of large-scale single-machine scheduling problems*, INFORMS Journal on Computing **17** (2005), 183–191.

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

[4] K. Bülbül, P. Kaminsky, and C. Yano, *Preemption in single machine earliness/tardiness scheduling*, Working paper, 2004.

[5] P.-C. Chang, *A branch and bound approach for single machine scheduling with earliness and tardiness penalties*, Computers & Mathematics with Applications **37** (1999), 133–144.

[6] H. Chen, C. Chu, and J.M. Proth, *An improvement of the Lagrangean relaxation approach for job shop scheduling: A dynamic programming method*, IEEE Transactions on Robotics and Automation **14** (1998), 786–795.

[7] J.Y. Chen and S.F. Lin, *Minimizing weighted earliness and tardiness penalties in single-machine scheduling with idle time permitted*, Naval Research Logistics **49** (2004), 760–780.

[8] N. Christofides, R. Alvarez-Valdes, and J.M. Tamarit, *Project scheduling with resource constraints: A branch and bound approach*, European Journal of Operational Research **29** (1987), 262–273.

[9] C. Chu, *A branch and bound algorithm to minimize total tardiness with different release dates*, Naval Research Logistics **39** (1992), 265283.

[10] J. Du and J.Y.T. Leung, *Minimizing total tardiness on one machine is np-hard*, Mathematics of Operations Research **15** (1990), no. 3, 483–495.

[11] B. Esteve, C. Aubijoux, A. Chartier, and V. Tkindt, *A recovering beam search algorithm for the single machine just-in-time scheduling problem*, European Journal of Operational Research (2005), in press.

[12] T.D. Fry, R.D. Armstrong, K. Darby-Dowman, and P.R. Philipoom, *A branch and bound procedure to minimize mean absolute lateness on a single processor*, Computers & Operations Research **23** (1996), 171–182.

[13] T.D. Fry, G. Leong, and T. Rakes, *Single machine scheduling: a comparison of two solution procedures*, Omega **15** (1987), 277–282.

[14] V. Gordon, J.M. Proth, and C. Chu, *A survey of the state-of-the-art of common due date assignment and scheduling research*, European Journal of Operational Research **139** (2002), 1–25.

[15] Refael Hassin and Mati Shani, *Machine scheduling with earliness, tardiness and non-execution penalties*, Computers & Operations Research **32** (2005), 683–705.

[16] Y. Hendel and F. Sourd, *Efficient neighborhood search for just-in-time scheduling problems*, European Journal of Operational Research (2005), to appear.

[17] J.A. Hoogeveen, *Multicriteria scheduling*, European Journal of Operational Research **167** (2005), 592–623.

[18] J.A. Hoogeveen and S.L. van de Velde, *A branch-and-bound algorithm for single-machine earliness-tardiness scheduling with idle time*, INFORMS Journal on Computing **8** (1996), 402–412.

[19] A. Jouglet, Ph. Baptiste, and J. Carlier, *Branch-and-bound algorithms for total weighted tardiness*, Handbook of Scheduling: Algorithms, Models and Performance Analysis (J.Y.T. Leung, ed.), Chapman & Hall/CRC Computer & Information Science Series, 2004, Chapter 13.

[20] C.A. Kaskavelis and M.C. Caramanis, *Efficient Lagrangian relaxation algorithms for industry size job-shop scheduling problems*, IIE Transactions **30** (1998), 1085–1097.

[21] S. Kedad-Sidhoum, Y. Rios-Solis, and F. Sourd, *Lower bounds for the earliness-tardiness scheduling problem on single and parallel machines*, Working paper – `www.optimization-online.org`, October 2004.

[22] Y.D. Kim and C. Yano, *Minimizing mean tardiness and earliness in single-machine scheduling problems with unequal due dates*, Naval Research Logistics **41** (1994), 913–933.

[23] J.-B. Lasserre and M. Queyranne, *Generic scheduling polyhedra and a new mixed-integer formulation for single-machine scheduling*, Proceedings of the 2nd IPCO Conference, 1992.

[24] V. Lauff and F. Werner, *Scheduling with common due date, earliness and tardiness penalties for multimachine problems: A survey*, Mathematical and Computer Modelling **40** (2004), 637–655.

[25] C.F. Liaw, *A branch-and-bound algorithm for the single machine earliness and tardiness scheduling problem*, Computers & Operations Research **26** (1999), 679–693.

[26] P.B. Luh, D.J. Hoitomt, E. Max, and K.R. Pattipati, *Schedule generation and reconfiguration for parallel machines*, IEEE Transactions on Robotics and Automation **6** (1990), 687–696.

[27] R.H. Möhring, A.S. Schulz, F. Stork, and M. Uetz, *Solving project scheduling problems by minimum cut computations*, Management Science **49** (2003), 330–350.

[28] L. Péridy, E. Pinson, and D. Rivreau, *Using short-term memory to minimize the weighted number of late jobs on a single machine*, European Journal of Operational Research **148** (2003), 591–603.

[29] M. Queyranne and A.S. Schulz, *Polyhedral approaches to machine scheduling*, Technical Report 408-1994, Technische Universität Berlin, 1994.

[30] F. Sourd, *The continuous assignment problem and its application to preemptive and non-preemptive scheduling with irregular cost functions*, INFORMS Journal on Computing **16** (2004), 198–208.

[31] ———, *Dynasearch neighborhood for the earliness-tardiness scheduling problem with release dates and setup constraints*, Operations Research Letters (2005), to appear.

[32] ———, *Optimal timing of a sequence of tasks with general completion costs*, European Journal of Operational Research **165** (2005), 82–96.

[33] F. Sourd and S. Kedad-Sidhoum, *The one machine problem with earliness and tardiness penalties*, Journal of Scheduling **6** (2003), 533–549.

[34] J.P. De Sousa and L.A. Wolsey, *A time-indexed formulation of non-preemptive single-machine scheduling problems*, Mathematical Programming **54** (1992), 353–367.

[35] S. Tanaka, T. Sasaki, and M. Araki, *A branch-and-bound algorithm for the single-machine weighted earliness-tardiness scheduling problem with job independent weights*, IEEE International Conference on Systems, Man and Cybernetics, 2003, pp. 1571–1577.

[36] J.M.S. Valente and R.A.F.S. Alves, *An exact approach to early/tardy scheduling with release dates*, Computers & Operations Research (2005), 2905–2918.

[37] S. Verma and M. Dessouky, *Single-machine scheduling of unit-time jobs with earliness and tardiness penalties*, Mathematics of Operations Research **23** (1998), 930–943.

[38] G. Wan and B.P.C. Yen, *Tabu search for single machine scheduling with distinct due windows and weighted earliness/tardiness penalties*, European Journal of Operational Research **142** (2002), 271–281.