Journal of Scheduling **6:** 533–549, 2003.

THE ONE-MACHINE PROBLEM WITH EARLINESS AND TARDINESS PENALTIES*

FRANCIS SOURDSAFIA KEDAD-SIDHOUM†

Laboratoire LIP6, 4, place Jussieu, 75 252 Paris, Cedex 05, France

ABSTRACT

We address the one-machine problem in which the jobs have distinct due dates, earliness costs, and tardiness costs. In order to determine the minimal cost of such a problem, a new lower bound is proposed. It is based on the decomposition of each job in unary operations that are then assigned to the time slots, which gives a preemptive schedule. Assignment costs are defined so that the minimum assignment cost is a valid lower bound. A branch-and-bound algorithm based on this lower bound and on some new dominance rules is experimentally tested.

KEY WORDS: one machine schedulingearliness/tardinesslower boundstransportation problems

1. INTRODUCTION

A set of jobs $\mathcal{J} = \{1, 2, \ldots, n$ has to be scheduled on a single machine. Each job $i \in \mathcal{J}$ has a *processing time* denoted by p_i, which is assumed to be integer. A *schedule* is a sequence of n *starting times* $S_1, S_2, \ldots, S_n$ subject to the constraints $S_i \geq S_j + p_j$ or $S_j \geq S_i + p_i$ (for all $i \neq j$). These constraints are called the *disjunctive constraints*. They mean that two jobs cannot be processed at the same time by the machine. Then a job i is scheduled—without preemption—from S_i to its completion time, denoted by C_i, equal to $S_i + p_i$. A cost function f_i is attached to each job $i \in \mathcal{J}$, the cost of job i in a schedule is $f_i(C_i)$. The *total cost* of the schedule is simply the sum of the costs of all the jobs, that is $\sum_{i \in \mathcal{J}} f_i(C_i)$. The problem is to find a schedule with a minimum cost.

A lower bound for this problem has recently been proposed by Rivreau (1999). Experimentally, it has been proved to be good for several classes of problems, including those of minimizing the weighted sum of tardiness and the weighted number of late jobs. In this paper, we will be more specially interested in the problem in which each cost function $f_i(C_i)$ is equal to $\max(\alpha_i(d_i - C_i), \beta_i(C_i - d_i))$. d_i represents the due date of job i: when i completes at time d_i, its cost is null and i is said to be *on-time*. If $C_i < d_i$, job i is *early* and its cost is $\alpha_i(d_i - C_i)$. α_i is the *earliness penalty of job i per unit time*. Symmetrically, if $C_i > d_i$, job i is *late*, its cost is $\beta_i(C_i - d_i)$, β_i being its *tardiness penalty per unit time*.

Several special cases of this problem, which is NP-complete even if there are no earliness penalties (Lenstra, 1977), have been studied. For instance, Garey, Tarjan, and Wilfong (1988) have presented an $O(n \log n)$ algorithm that determines an optimal schedule for identical

* This work has been partially financed by ILOG S.A., under research contract ILOG/UPMC number: 000320
† Correspondence to: Safia Kedad-Sidhoum, E-mail: Safia Kedad-Sidhoum@lip6.fr

processing times ($p_i = p$) and unary earliness and tardiness penalties ($\alpha_i = \beta_i = 1$). When the processing times are equal, the problem becomes an assignment problem if $p_i = 1$ and d_i integer (Lawler, 1976). If $p_i = p$, Verma and Dessouky (1998) have shown that other subproblems (including $\alpha_i = \beta_i$) can be solved by a LP-relaxation that renders an integral solution but, to our knowledge, the complexity of the general problem with $p_i = p$ is still an open question. Much attention has also been paid to the common due-date case ($d_i = d$) for which there exists a strong characterization of a class of optimal solutions Bakker and Scudder (1990). Thanks to these properties, the problems with up to 125 jobs can be solved at optimality Van den Akker, Hoogeven, and van de Velde (1998). Unfortunately, these properties cannot be adapted for the general case. Hoogeveen and van de Velde (1996) presented a branch-and-bound algorithm for the problem with asymmetric but job-independent earliness–tardiness penalties (i.e., $\alpha_i = \alpha$ and $\beta_i = \beta$) but they concluded that, due to the difficulty to compute strong lower bounds, some problems with 15 jobs cannot even be solved.

We naturally had to face the same difficulty to compute a lower bound for our problem with job dependent earliness–tardiness penalties. In our approach, we gave the greatest importance to the quality of the lower bound even if its computation time may be long. The lower bound we present can be computed in pseudo-polynomial time $O(n^2T)$ where T is the time horizon of the schedule. In fact, this bound is valid for general start-dependent cost functions (i.e., not only for earliness–tardiness costs), assuming that the task start times are integers. Moreover, this bound can be used to derive very good feasible schedules and efficient branching rules in a branch-and-bound approach.

Section 2 presents the lower bound and an algorithm to compute it. In Section 3, we show how the lower bound can be integrated into a branch-and-bound algorithm, that is how it can be used to compute lower bounds at each node in presence of a partial schedule. Some dominance rules and a branching scheme based on the information given by the bound are also presented. Computational results are given in Section 4.

2. AN ASSIGNMENT-BASED LOWER BOUND

2.1. Formulation of LB

In this section, we make no assumption on the form of the cost functions f_i. However, we need two necessary additional assumptions for the lower bound to be valid:

1. There exists an optimal *integer schedule*, that is an optimal schedule in which all the start times are integer. Note that when a problem does not satisfy this assumption, a lower bound can nevertheless be derived by replacing the cost functions f_i by the "lower step functions" $\bar{f}_i(C_i) = \min_{\lfloor C_i \rfloor \le t < \lceil C_i \rceil} f_i(t)$. Indeed, an instance with the cost functions $\bar{f}_i$ has an integer optimal schedule and this schedule has a cost lower than the optimum of the instance with the cost functions f_i.
2. All the tasks must be scheduled between the origin 0 and the horizon T. T must be finite since the time complexity of the proposed algorithm will depend on T.

The lower bound is based on the decomposition of each job into unary operations that are then assigned to the available time slots. So we define $\mathcal{T} - \{1, 2, \ldots, T\}$ as the set of the time slots. The integer $t \in \mathcal{T}$ represents the time interval $[t-1, t)$. Each job i is divided into p_i unit

execution time (UET) operations $o_{i1}, o_{i2}, \ldots, o_{ip_i}$. The operation o_{ij} with $1 \leq i \leq n$ and $1 \leq j \leq p_i$ is said to be the jth operation of the job i. Let $\mathcal{O}$ be the set of the operations ($|\mathcal{O}| = \sum_{i \in \mathcal{J}} p_i$).

We now define the assignment costs in order to have the following property: when all the operations of a given job i are assigned to p_i adjacent time slots, the total assignment cost of the p_i unary operations must be equal to the cost of scheduling the job i in the corresponding time interval. More formally, for each job i and each slot t, let c_{it} be the cost that corresponds to the assignment of any operation of i to the time slot t. The costs are operation-independent, that is two operations o_{ij} and $o_{ij'}$ ($j \neq j'$) of the same job have the same cost c_{it}. We want the costs c_{it} to satisfy the following equations:

$$\sum_{t \leq t' < t+p_i} c_{it'} = f_i(t + p_i) \quad \forall t \in [1,\ T + 1 - p_i],\ \forall i \in \mathcal{J} \tag{1}$$

In this system of linear equations, for each job i, T cost values $c_{i1}, \ldots, c_{iT}$ have to be determined subject to $T + 1 - p_i$ independent linear equations. As a consequence, all the costs c_{it} are fixed as soon as the costs are fixed for $p_i - 1$ distinct values of t. In particular, if the processing time of the job i is $p_i = 1$, the cost for i at each time t is $c_{it} = f_i(t + 1)$. Arbitrarily, we will fix for each job $i \in \mathcal{J}$ the cost of the first $p_i - 1$ slots by the so-called *initial conditions*:

$$\forall i \in \mathcal{J} \quad \text{and} \quad 1 \leq t < p_i,\ \ c_{it} = \lambda_{it} \tag{2}$$

For the sake of shorter notations, we define the set Ω of all the possible indices for the *initial values* λ_{it}:

$$\Omega = \bigcup_{i \in \mathcal{J}} \{(i, 1), \ldots, (i, p_i - 1)\}$$

So, all the initial values can be seen as a vector $\lambda \in \mathrm{R}^{\Omega}$:

$$(\lambda_{1,1}, \ldots, \lambda_{1,p_1-1}, \lambda_{2,1}, \ldots, \lambda_{i,p_i-1}, \lambda_{i+1,1}, \ldots, \lambda_{n,p_n-1})$$

Note that R^{Ω} is equivalent to the space vector of dimension $|\Omega| = (\sum_{i \in \mathcal{J}} p_i) - n$. For any $\lambda \in \mathrm{R}^{\Omega}$, c^{λ}_{it} will denote the solution of the Equation (1) submitted to the initial conditions (Equation (2)). It can be easily shown that for any $t \in \mathcal{T}$ and $i \in \mathcal{J}$:

$$c^{\lambda}_{it} = \begin{cases} c^0_{it} - \sum_{j=1}^{j=p_i-1} \lambda_{ij} & \text{if } t_i = 0 \\ c^0_{it} + \lambda_{it_i} & \text{otherwise} \end{cases} \tag{3}$$

where t_i is the integer in $[0, p_i - 1]$ such that $t_i \equiv t$ modulo p_i. c^0_{it} it is the cost c_{it} defined for the null vector $\lambda = (0, 0, \ldots, 0)$. For any vector $\lambda \in \mathrm{R}^{\Omega}$, we consider the complete weighted bipartite graph $G(\lambda)$ between sets $\mathcal{O}$ and $\mathcal{T}$ where the cost of arc $(o_{ij}, t) \in \mathcal{O} \times \mathcal{T}$ is c_{it}. Let $\mathrm{LB}(\lambda)$ be the minimal cost of a maximum matching in $G(\lambda)$. Since the graph is complete and $|\mathcal{T}| \leq |\mathcal{O}|$, such a maximum matching has a cardinality $|\mathcal{O}| = \sum_{i \in \mathcal{J}} p_i$.

Theorem 1. For any $\lambda \in \mathrm{R}^{\Omega}$, $\mathrm{LB}(\lambda)$ *is a lower bound for the one-machine problem with general cost functions.*

Proof. Let us consider a feasible schedule for the one-machine problem with general cost functions. From this schedule, we can build a maximum matching in the complete bipartite graph between $\mathcal{O}$ and $\mathcal{T}$ by assigning o_{ij} to $S_i + j - 1$. The cost of such a matching is $\sum_{i \in \mathcal{J}} \sum_{1 \leq j < p_i} c_{i,S_i+j-1}$. Equation (1) show that this cost is $\sum_{i \in \mathcal{J}} f_i(S_i + p_i)$ that is the cost of the

schedule for the one-machine problem with general cost function. As a consequence, whatever the value of λ is, LB(λ) is a lower bound for this problem. ■

Since LB(λ) is a lower bound of our problem for any vector λ, we want to have a λ that maximizes LB(λ) in order to have the best possible lower bound. We first present how to compute LB(λ) for a given λ.

2.2. *An algorithm for computing* LB(λ)

In all this section, λ is assumed to be fixed and the costs c_{it}^{λ} it will be simply denoted by c_{it}.

Computing LB(λ) is a well-known problem, namely the *assignment problem*. It consists in finding the minimum-cost maximum matching between $\mathcal{O}$ and $\mathcal{T}$. This problem is very classic and several algorithms exist to solve them. They are reviewed in a recent paper of Dell'Amico and Toth (2000) where codes are experimentally compared. Most of them are based on a primal-dual approach and the most efficient algorithms to solve this problem run in $O(T^3)$. In our assignment problem, the p_i UET operations from the same job i have the same costs. Hence, the problem can be seen as a *transportation problem* in which each plant $i \in \mathcal{J}$ offers p_i unit of goods and each demand $t \in \mathcal{T}$ is unary. The transportation cost from i to t is c_{it}. We did not find in the literature any algorithm for this particular class of transportation problems and we propose an $O(n^2T)$ algorithm that is an adaptation of the famous Hungarian algorithm (kuhn, 1955; Lawler, 1976).

The Hungarian algorithm requires a perfect matching. So, $p_0 = |\mathcal{T}| - |\mathcal{O}|$ UET operations have to be added in order to have as many operations as slots: let $o_0, \ldots, o_{0,p_0}$ be these operations that will be called the *idle operations* in the sequel. These operations will be considered as the partition of a virtual job of processing time p_0 called the *idle job* and denoted by the index 0. Let $\bar{\mathcal{O}}$ be the set of all the—idle and non-idle—operations (we now have that $|\bar{\mathcal{O}}| = |\mathcal{T}| = T$) and let $\bar{\mathcal{J}} = \{0\} \cup \mathcal{J}$ be the set of all the jobs. The cost of the idle operations c_{0t} is set to a constant M for any time t. This constant M is assumed to be larger than any other cost c_{it} for $i \in \mathcal{J}$ and $t \in \mathcal{T}$. When a perfect matching between $\bar{\mathcal{O}}$ and $\mathcal{T}$ is found, LB(λ) is the sum of the costs of the non-idle operations assigned in this matching.

2.2.1. *Augmenting paths*

The key of decreasing the $O(T^3)$ complexity of the Hungarian algorithm to an $O(n^2T)$ algorithm for our problem is of course that the p_i operations coming from the same job i are identical. For that, we extend the definition of some usual terms for the assignment problem (see e.g. Khuller and Raghavachari (1999)) so that they match the transportation problem between $\bar{\mathcal{J}} = \{0\} \cup \mathcal{J}$ and $\mathcal{T}$. A subset $\mathcal{M}$ of $\bar{\mathcal{J}} \times \mathcal{T}$ will be called a $\bar{\mathcal{J}}$-*matching* between $\bar{\mathcal{J}}$ and $\mathcal{T}$ if, for any $i \in \bar{\mathcal{J}}, d_i^+(\mathcal{M}) \leq p_i$ and, for any $t \in \mathcal{T}, d_t^-(\mathcal{M}) \leq 1$. $d_t^+(\mathcal{M})$ and $d_t^-(\mathcal{M})$ are respectively the degree of i and t in M and are defined as: $d_t^+(\mathcal{M}) = |\{(i,t) \in \mathcal{M}, t \in \mathcal{T}\}|$ and $d_t^-(\mathcal{M}) = |\{(i,t) \in \mathcal{M}, t \in \bar{\mathcal{J}}\}|$. With this definition, and if each operation $o_{ij} \in \bar{\mathcal{O}}$ is considered as a job with a processing time $p_{ij} = 1$, an $\bar{\mathcal{O}}$-matching corresponds to the usual definition of a matching. Given a $\bar{\mathcal{J}}$-matching $\mathcal{M}$ we can build the $\bar{\mathcal{O}}$-matching $\hat{\mathcal{M}}$:

$$\hat{\mathcal{M}} = \{(o_{ij}, t) \in \bar{\mathcal{O}} \times \mathcal{T} | (i,t) \in \mathcal{M} \quad \text{and} \quad j \leq d_i^+(\mathcal{M})\}$$

$\hat{\mathcal{M}}$ is called the $\bar{\mathcal{O}}$-matching associated to $\mathcal{M}$. Clearly, $\mathcal{M}$ and $\hat{\mathcal{M}}$ have the same size. c_{it} being the cost of the edge (i,t) in a $\bar{\mathcal{J}}$-matching and of the edge (o_{ij}, t) in a $\bar{\mathcal{O}}$-matching, $\mathcal{M}$ and $\hat{\mathcal{M}}$ have also the same cost. A $\bar{\mathcal{J}}$-matching $\mathcal{M}$ is *perfect* if, for any $i \in \bar{\mathcal{J}}, d_i^+(\mathcal{M}) = p_i$, and, for any $t \in \bar{\mathcal{T}}, d_t^-(\mathcal{M}) = 1$. If $\mathcal{M}$ is perfect, $\hat{\mathcal{M}}$ is also perfect. When talking about properties valid for

both $\mathcal{M}$ and $\hat{\mathcal{M}}$, we will simply say a matching instead of $\bar{\mathcal{J}}$- and $\bar{\mathcal{O}}$-matching. For example: "A matching is perfect if and only if its size is T."

The classic dual-based results for the assignment problem can be found in Khuller and Raghavachari (1999) and Nemhauser and Wolsey (1988). By adapting these results, we know that any optimal $\bar{\mathcal{J}}$-matching is a $\bar{\mathcal{J}}$-matching in a graph G_{uv} for some $u=(u_0, u_1, \ldots, u_n)$ and $v=(v_1, \ldots, v_T)$ such that, for any $i \in \bar{\mathcal{J}}$ and $t \in \mathcal{T}$, $u_i + v_t \leq c_{it}$ and G_{uv} is the bipartite graph between $\bar{\mathcal{J}}$ and $\mathcal{T}$ that contains the edges (i, t) satisfying the equality $u_i + v_t = c_{it}$. u_i and v_t are the dual values and $u_i + v_t \leq c_{it}$ are the dual constraints. $\hat{G}_{uv}$ is naturally the bipartite graph between $\bar{O}$ and $\mathcal{T}$ that contains the edges (o_{ij}, t) such that $u_i + v_t = c_{it}$. Subsequently, the algorithm solves simultaneously these two problems: finding a maximum matching in a graph and finding the values u_i and v_t such that G_{uv} contains a perfect matching. It starts with the empty matching $\mathcal{M} =$ and with $u_i = 0$ and $v_t = \min\{c_{it} | i \in \mathcal{J}\}$ then, at each step, either augments the matching or modifies the dual variables. More precisely, at each step, the algorithm tests whether the current matching $\mathcal{M}$ is maximum for the current G_{uv}. If it is not maximum, then $\mathcal{M}$ is augmented else u_i and v_t are modified so that $\mathcal{M}$ is still a matching of G_{uv} but not a maximum matching. Therefore, the algorithm goes on augmenting $\mathcal{M}$ for the new subgraph. The algorithm stops when a perfect matching is found.

We now introduce some additional definitions that are again related to the classical assignment problem vocabulary. An edge of G_{uv} is *matched* if it is in $\mathcal{M}$ else it is *free*. A vertex $i \in \bar{\mathcal{J}}$ (resp. $t \in \mathcal{T}$) is *matched* if $d_i^+(\mathcal{M}) = p_i$(resp. $d_t^-(\mathcal{M}) = 1$). A vertex that is not matched is *free*. A path $\mu = (J_1^\mu, T_1^\mu, \ldots, J_{k-1}^\mu, T_{k-1}^\mu, J_k^\mu)$ or $\mu = (J_1^\mu, T_1^\mu, \ldots, J_k^\mu, T_k^\mu)$—with $\forall i$, $J_i^\mu \in \bar{\mathcal{J}}$, and $T_i^\mu \in \mathcal{T}$ and $\forall i \neq j, J_i^\mu \neq J_j^\mu$, and $T_i^\mu \neq T_j^\mu$ is called an *alternating path* (with respect to $\mathcal{M}$) if the edges (J_j^μ, T_j^μ) are free and if the edges (T_j^μ, J_{j+1}^μ) are matched. An *augmenting path* (with respect to $\mathcal{M}$) is an alternating path in which the first and the last vertices are free. An augmenting path contains an even number of vertices so an odd number of edges. If we have an augmenting path in G_{uv}, we can easily build an augmenting path (with the usual meaning) in $\hat{G}_{uv}$. Conversely, if we have an augmenting path in $\hat{G}_{uv}$, we can also construct an augmenting path in G_{uv} by removing the parts of the path between operations of the same job:

Lemma 2. There is an augmenting path in $\hat{G}_{uv}$ with respect to $\hat{\mathcal{M}}$, if and only if there is an augmenting path in G_{uv} with respect to $\mathcal{M}$.

The symmetric difference of a matching $\mathcal{M}$ and an augmenting path μ, denoted by $\mathcal{M} \oplus \mu$, is defined to be the set $(\mathcal{M} \backslash E(\mu)) \cup (E(\mu) \backslash \mathcal{M})$ where $E(\mu)$ is the set of the edges in the path μ: $E(\mu) = \{(J_1^\mu, T_1^\mu), (T_1^\mu, J_2^\mu), \ldots\}$. Clearly, $\mathcal{M} \oplus \mu$ is also a matching and $|\mathcal{M} \oplus \mu| = |\mathcal{M}| + 1$. So, if there is an augmenting path in G_{uv} with respect to $\mathcal{M}$, $\mathcal{M}$ is not maximum. Conversely, if $\mathcal{M}$ is maximum in G_{uv}, $\hat{\mathcal{M}}$ is maximum in $\hat{G}_{uv}$ so that there is no augmenting path for $\hat{\mathcal{M}}$. Lemma 2 shows that there is no augmenting path for $\mathcal{M}$ either. We have then extended for the $\bar{\mathcal{J}}$-matching the well-known necessary and sufficient condition for a matching (in the ordinary sense) to be maximum:

Theorem 3. A $\bar{\mathcal{J}}$-matching $\mathcal{M}$ in G_{uv} is a maximum matching if and only if there is no augmenting path in G_{uv} with respect to $\mathcal{M}$.

Theorem 3 is important because the augmenting paths in G_{uv} have at most $2n$ vertices whereas the augmenting paths in $\hat{G}_{uv}$ have up to $2T$ vertices. However, the graph G_{uv} has $n + T$ vertices and $O(nT)$ edges so that we need specific data structures to find these augmenting paths.

2.2.2. Data structures

The algorithm uses data structures that can be initialized in $O(nT)$ time. With these data structures, we can find in $O(n^2)$ time whether a matching can be augmented. If the matching is augmented, the data structures are updated in $O(n^2)$ time, otherwise, the dual values are modified in $O(nT)$ time and the data structures are updated with the same complexity. Finally, it will be shown that the matching is augmented T times and the dual values are modified at most n times to prove that the algorithm runs in $O(n^2T)$ time. The two main data structures are called $\mathcal{F}$ and $\mathcal{E}$. $\mathcal{F}$ is a one-dimensional array of lists (one list for each job in $\bar{\mathcal{J}}$) and $\mathcal{E}$ is a two-dimensional array of lists (one for each pair of jobs in $\bar{\mathcal{J}}$). Any list $\mathcal{F}_i$ or $\mathcal{E}_{ij}$ will contain a subset of $\mathcal{T}$. Two basic data structures are used to store it: a (non-sorted) doubly-linked list that contains the time slots and an array of T pointers. The pointer in position i in the array indicates the presence of the ith time slot in the list: it is either null or points to the location of the element in the list. The size of such a structure is clearly $O(T)$. The presence of a time slot can be tested in constant time, an element can be inserted, found, and removed in constant time. $\mathcal{F}$ and $\mathcal{E}$ requires, respectively $O(nT)$ and $O(n^2T)$ space.

$\mathcal{F}_i$ is used to store all the time slots t such that t is a free vertex and (i, t) is an edge of G_{uv}. $\mathcal{E}_{ij}$ is used to store all the time slots t such that $(i, t) \in \mathcal{M}$ and (j, t) is an arc of G_{uv}, in which case (i, t) is an arc of G_{uv} and $(j, t) \notin \mathcal{M}$.

The matching $\mathcal{M}$ is represented by an array of T elements, denoted by the sequence $(\mathcal{M}_t)_{t\in\mathcal{T}}$, of T elements. $\mathcal{M}_t$ is -1 if t is free and $\mathcal{M}_t$ is equal to i if $(i, t) \in \mathcal{M}$. The last data structure maintained by our algorithm is an array that contains the degree $d_i^+(\mathcal{M})$ of each $i \in \bar{\mathcal{J}}$. When the algorithm starts, $\mathcal{M} = \emptyset$ so that all these structures can be initialized in $O(n^2T)$ time.

2.2.3. A step of the algorithm

Let us now consider in detail a step of the algorithm. If the size of the current matching $\mathcal{M}$ is T the matching is perfect and the algorithm can stop returning $\mathcal{M}$. Otherwise, a possible augmenting path is searched for. In this goal, we define the graph $\mathcal{G}$ whose vertex set is $\bar{\mathcal{J}}$ and edges are the pairs $(i, j) \in \bar{\mathcal{J}}^2$ for which the list $\mathcal{E}_{ij}$ is not empty. Let $F \subset \mathcal{J}$ be the set of the tasks i such that $\mathcal{F}_i$ is not empty and let $F' \subset \bar{\mathcal{J}}$ be the set of the tasks i such that $d_i^+(\mathcal{M}) < p_i$. Clearly, if there is $i \in F \cap F'$, the edge (i, t), for any $t \in \mathcal{F}_i$, can be added to the matching—the edge (i, t) is an augmenting path. If $F \cap F' = \emptyset$, we search for a path from F to F' in the graph $\mathcal{G}$. If such a path $\rho = (\rho_1, \rho_2, \ldots, \rho_p)$ exists, we can build an augmenting path $\mu = (J_p^\mu, T_p^\mu, \ldots, J_2^\mu, T_2^\mu, J_1^\mu, T_1^\mu)$ where

- for any $1 \le i \le p$, $J_i^\mu = \rho_i$
- for any $1 \le i \le p$, T_i^μ is any time slot in the list $\mathcal{E}_{\rho_{i-1}\rho i}$
- T_1^μ is any time slot in $\mathcal{F}_{\rho_1}$.

By construction of $\mathcal{G}$, the list in which T_i^μ is chosen cannot be empty. Since J_i^μ is free $T_1^\mu \neq T_j^\mu$ for any $j > 1$ and, for any $1 < i < j \le p$, $T_i^\mu \neq T_j^\mu$ because T_j^μ is matched to job J_{j-1}^μ whereas T_i^μ is matched to job J_{i-1}^μ. So the sequence μ is a path and this path is alternating. This is an augmenting path because T_1^μ is free and $d_{\rho_p}^+(\mathcal{M}) < p_{\rho_\rho}$ (by construction). Conversely, it can be easily shown that there is an alternating path in G_{uv} with respect to $\mathcal{M}$ if and only if there is a path from F to F' in $\mathcal{G}$. By exploring $\mathcal{G}$ from F it is easy to find a path from F to F', that is an augmenting path in G_{uv}—or to prove that there is no such path—in $O(n^2)$ time (see Theorem 3). We now show how the structures are updated whether there exists an augmenting path or not.

There is an augmenting path. Let $\mu = (J_p^\mu, T_p^\mu, \ldots, J_2^\mu, T_2^\mu, J_1^\mu, T_1^\mu)$ be the augmenting path found in G_{uv} by our algorithm. We have proven in Section 2.2.1 that $p \leq n$. So the matching array $\mathcal{M}$ can be updated in $O(n)$ time. The degree of J_p^μ, namely $d^+_{\rho_\rho}(\mathcal{M})$, must be increased by one. The time slot T_1^μ is not free anymore so that it must be removed from each list $\mathcal{F}_i$ in which it was present. From Section 2.2.2, this can be done in constant time for each list so that $\mathcal{F}$ is updated in $O(n)$ time. For each j such that $1 < j \leq p$, the edge (J_j^μ, T_{j-1}^μ) is removed from the matching so that for any $i \in \bar{\mathcal{J}}$ such that (i, T_j^μ) is an edge of G_{uv}, the time slot T_{j-1}^μ must be removed from the list $\mathcal{E}_{\rho_j i}$. For each j such that $1 \leq j \leq p$, the edge (J_j^μ, T_j^μ) must be added to the matching so that for any $i \in \bar{\mathcal{J}}$ such that (i, T_j^μ) is an edge of G_{uv}, the time slot T_j^μ must be added to the list $\mathcal{E}_{\rho_j i}$. Then, $\mathcal{E}$ is updated in $O(n^2)$ time—see Theorem 3.

There is no augmenting path. Let $\bar{F}$ be the set of jobs that can be reached from F by a direct path in $\mathcal{G}$. F of course includes F. This set has been determined when searching for an augmenting path. Let $\bar{\mathcal{T}}$ be the set of the free time slots and the time slots matched to a job in $\bar{F}$. We now determine

$$\delta = \min_{i \in \bar{\mathcal{J}} - \bar{F}; t \in \bar{\mathcal{T}}} c_{it} - u_i - v_t$$

which can be done in $O(nT)$ time. With respect to the dual constraints, we will have $\delta > 0$. The dual variables are modified by decreasing u_i by δ for each $i \in \bar{F}$ and increasing v_t by δ for each $t \in \bar{\mathcal{T}}$. Obviously, the edges between $\bar{F}$ and $\bar{\mathcal{T}}$ and between $\bar{\mathcal{J}} - \bar{F}$ and $\mathcal{T} - \bar{\mathcal{T}}$ are not modified by the modification of the dual variables. Some (at least one) edges are added between $\bar{\mathcal{J}} - \bar{F}$ and $\bar{\mathcal{T}}$ and some edges are removed between $\bar{F}$ and $\mathcal{T} - \bar{\mathcal{T}}$. These modifications are made in $O(nT)$ time. Since there were no edges of the matching $\mathcal{M}$ between $\bar{F}$ and $\mathcal{T} - \bar{\mathcal{T}}$, the current matching is still a matching for the new graph G_{uv}. For each edge (i, t) removed from G_{uv}, the time slot t is in $\mathcal{T} - \bar{\mathcal{T}}$ so that this slot t is matched and then must be removed from $\mathcal{E}_{i,\mathcal{M}_t}$. For each edge (i, t) added to G_{uv} by the modification of the dual values, the slot t is either free in which case t is added to $\mathcal{F}_i$ or matched in which case t is added to $\mathcal{E}_{i,\mathcal{M}_t}$. Since, at most nT arcs are added or removed, $\mathcal{F}$ and $\mathcal{E}$ are updated in $O(nT)$ time.

2.2.4. Complexity

When an augmenting path is found, the cardinality of the current matching is increased by one (see Theorem 3). $|\mathcal{M}|$ is not modified when there is no augmenting path in G_{uv}. Therefore, there are at most T steps in which an augmenting path is found. When there is no augmenting path in G_{uv} with respect to the current matching $\mathcal{M}$, at least one new edge (i, t) is added to G_{uv} between $\bar{\mathcal{J}} - \bar{F}$ and $\bar{\mathcal{T}}$. It is easy to see that when $\mathcal{F}$ and $\mathcal{E}$ are updated, i is added either to F or an arc between a job in $\bar{F}$ and i is created in $\mathcal{G}$. So $\bar{F}$ has at least one new element. When an augmenting path is found, G_{uv} and $\bar{F}$ are not modified. Therefore, there are at most n steps in which there is no augmenting path in G_{uv}. So there are at most $n + T$ steps in the algorithm and its time complexity is $O((n+T) \times n^2 + T \times n^2 + n \times (nT))$. Since $T > n$, the time complexity is $O(n^2 T)$. The space complexity is also $O(n^2 T)$ (see Section 2.2.2).

2.2.5. Computation heuristics

It is well known that the efficiency of the Hungarian algorithm greatly depends on two heuristics (Dell'Amico and Toth, 2000): one for building the initial assignment and one for finding more quickly the augmenting paths. Since this algorithm is used in a branch-and-bound procedure, the initial dual partial solution can be built from the optimal dual solution of the

ascendant node in the search tree. Indeed, we will see in Section 3 that between a node and its ascendant in the search tree, there are only few differences: some costs c_{it} are modified. Therefore, the heuristic decreases the value v_t of the optimal solution of the ancestor each time the dual constraint $u_i + v_t \leq c_{it}$ is violated in the descendant problem.

A second implementation trick comes from the observation that several operations of a same task are often scheduled in adjacent slots. So, we implemented $\mathcal{F}_i$ and $\mathcal{E}_{ij}$ as a list of intervals of slots instead of a simple list of slots. So, when a path ρ from F to F' is found in $\mathcal{G}$, several "parallel" augmenting paths in G_{uv} can be derived from ρ and we can process the matching transformation using all these augmenting paths in the same step.

2.3. *Maximizing* LB(λ)

Section 2.2 was devoted to the computation of LB(λ) for some given vector λ. In order to get the best possible lower bound, we would like to maximize the function LB(λ). Annex A presents the link between LB(λ) and the Lagrangean of an integer programing formulation of the problem. Therefore, LB(λ) can be maximized through a subgradient optimization scheme. However, we were unable to have a fast convergence to the maximum of LB(λ).

An alternative approach that seems better for our branch-and-bound algorithm was to heuristically select a problem-dependent λ^* such that LB(λ^*) is a good lower bound. In Section 4, we present how we built such a good vector for the earliness–tardiness problem.

3. THE EARLINESS–TARDINESS PROBLEM

In the remaining sections of this paper, we consider the earliness–tardiness problem, where the cost due to the completion of task i at time C_i is $f_i(C_i) = \max(\alpha_i(d_i - C_i), \beta_i(C_i - d_i))$. We assume that all the due dates d_i and all the processing times p_i are integers. We will show (in Section 3.1) that this assumption implies there is an optimal schedule in which all the start times are integers. Furthermore, we can easily find an integer T such that there is an optimal schedule that completes at or before T. So, the lower bounds presented in Section 1 can be directly applied to the earliness–tardiness problem.

The results presented in this section remain valid when the cost functions f_i are simply assumed to be convex, provided we know there is an integer optimal schedule. When the functions are convex and piecewise linear, we can show that a sufficient condition to have an integer optimal schedule is that all the irregular points of the functions f_i has an integer abscissa. This includes the practical case of earliness–tardiness scheduling with a due interval in which the job is on time when it completes in a given interval (instead of at a sharp time).

3.1. *Optimal schedule of a chain of tasks*

In this section, we describe a procedure to find an optimal schedule when the operations are constrained to be scheduled in a given order, say $1, 2, \ldots, n$. This procedure will be intensively used in the branch and bound algorithm so that we present it with some details.

An $O(n \log n)$ algorithm has been developed by Garey, Tarjan, and Wilfong (1988) and shown to solve the problem with $\alpha_i = \beta_i = 1$. Hoogeveen and van de Velde (1996) noticed that this algorithm also solves the problem with $\alpha_i = \alpha$ and $\beta_i = \beta$ and Chrétienne (1998) and Chrétienne and Sourd (2002) extend the proof for the general case. They also prove that the algorithm

works when the cost function f_i of each operation i is convex. Here we just give a short description of the procedure. For more details and for the proof of the algorithm, we refer to Chrétienne and Sourd (2002).

The key property that funds the algorithm is that the cost of scheduling the jobs $k, k+1, \ldots, k'$ in a block, that is without idle time in between them, is a piecewise linear convex function. The algorithm inserts the tasks in the schedule in the order of the given sequence. The first operation is scheduled so that it completes at time d_1. We now assume that $S_1^k, \cdots, S_k^k$ is an optimal schedule for the first k tasks and an optimal schedule $S_1^{k+1}, \ldots, S_k^{k+1}, S_{k+1}^{k+1}$ for the first $k+1$ tasks is searched for. If $S_k^k + p_k \leq d_{k+1} - p_{k+1}$, the task $k+1$ can complete at its due date while meeting the resource constraint. Hence, an optimal schedule is built. If $S_k^k + p_k > d_{k+1} - p_{k+1}$, the task $k+1$ is added at the end of the last block of the schedule $S_1^k, \ldots, S_k^k$ and the block is left-shifted until the cost function of the block—which is convex—reaches its minimum or until the start time of the block becomes equal to the end time of the previous block or becomes equal to 0. If the block starts at time 0 or if its cost is minimum, the schedule is optimal. Otherwise the two last blocks are merged and the algorithm iterates the left-shifting process with the new block.

In what follows, we will need to compute, for a given $t \geq \sum_{i \in \mathcal{J}} p_i$, the minimal cost $\Sigma(t)$ of scheduling the chain of tasks so that the schedule completes before time t. $\Sigma(t)$ can also be computed in $O(n \log n)$ time as follows: assume that $S_1^n, \ldots, S_n^n$ is the optimal schedule for the problem without the additional constraint on the makespan, then we just have to shift the last block left (and eventually merge it with previous blocks) until it completes at t. In particular, it can easily be shown that an array containing $\Sigma(t)$ for each value of $t \in [\sum_{i \in \mathcal{J}} p_i, T]$ can be initialized in $O(T \log n)$ time. Notice that $\Sigma(t)$ is a nonincreasing and convex function which is constant for $t \geq S_n^n$.

When there is no constraint upon the makespan of the schedule, we know that, in an optimal schedule, each block is scheduled at its minimum cost. Since the cost function of the block B is $f_B(t) = \sum_{i \in B} f_i(t + \sum_{j \in B;\, j \leq i} p_i)$, we know that, in the earliness–tardiness problem, the minimum of this convex function is reached at one of the times $d_i - \sum_{j \in B;\, j \leq i} p_i$ for some $i \in B$—note that the function may be constant between two consecutive such times. Since all these times are integer, there exists an optimal schedule in which each block (and so each task) starts at an integer time. For any fixed sequence of tasks, there is a minimal schedule with integer start times so that there is also an optimal schedule in which each task has an integer start time.

In order to show how the lower bounds presented in Section 2 can be used, we just have to exhibit a horizon T such that there is at least one optimal schedule completing at or before T. For example, the value $T = \max_{i \in \mathcal{J}} d_i + \sum_{i \in \mathcal{J}} p_i$ is such a T. However, it should be interesting to have a better upper bound because the smaller the T, the faster the lower bounds are computed. Unfortunately, we were not able to derive a better horizon.

3.2. *Lower bounds for the cost of a partial schedule*

The lower bound presented in the Section 2 assumes no precedence constraints. In a branch-and-bound scheme, such a situation occurs only at the root of the enumeration tree. At any other node, the tasks are partially sequenced. In this section, we present how such information is used to compute the lower bound.

We consider the problem of scheduling the tasks in $\mathcal{J} = \{1, 2, \ldots, n\}$ in presence of a precedence graph G_k that represents the partial sequence (see Figure 1). G_k is assumed to have

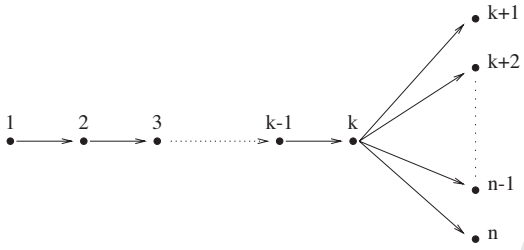

Figure 1. The graph G_k

the particular structure shown in Figure 1 that comes from the branching scheme. The tasks $1, \ldots, k$ are called the *sequenced tasks* while the tasks in $[k+1, n]$ are called the *non-sequenced tasks*.

Let $I = [\sum_{i=1}^{k} p_i, T - \sum_{i=k+1}^{n} p_i]$, for any $t \in I$, $\Sigma(t)$ denotes the minimum cost for scheduling the chain $1 \rightarrow \cdots \rightarrow k$ so that this schedule completes before t. Now $\mathrm{LB}(t)$ is the lower bound for scheduling the non-sequenced tasks $k+1, \ldots, n$ in the time interval $[t, T]$. Note that such a bound can easily be derived from the assignment-based lower bound presented in Section 1 by setting all the assignment costs $c_{it'}$ to the constant M for any $t' < t$.

3.2.1. Formulation of LB^+

Let us consider the schedule $S_1^\star, \ldots, s_n^\star$ that minimizes the total cost for problem with the precedence graph G_k. By definition of $\Sigma(t)$, we have:

$$\sum_{i=1}^{k} f_i(S_i^\star + p_i) \geq \Sigma(S_k^\star + p_k) \tag{4}$$

By definition of $\mathrm{LB}(t)$, we also have:

$$\sum_{i=k+1}^{n} f_i(S_i^\star + p_i) \geq \mathrm{LB}(S_k^\star + p_k) \tag{5}$$

Equations (4) and (5) show that $\Sigma(S_k^\star + p_k) + \mathrm{LB}(S_k^\star + p_k)$ is a lower bound for the cost of $S^\star$. Computing the value $S_k^\star$ is of course a hard problem but we know that $S_k^\star + p_k \in I$. As a consequence,

$$\mathrm{LB}^+ = \min_{t \in I} \Sigma(t) + \mathrm{LB}(t)$$

is a lower bound for the cost of any schedule subject to the precedence graph G_k. The computation of LB^+ is quite time-consuming because $\mathrm{LB}(t)$ must be computed many times. However, it is not necessary to compute $\mathrm{LB}(t)$ from scratch for each value of t. Indeed, when $\mathrm{LB}(t)$ is computed, we get a maximum matching with minimum cost between $\bar{\mathcal{O}} = \{o_{ij} \in \mathcal{O} | i \in [k+1, n]\}$ and $[t, T]$. In order to have a minimum cost matching between $\bar{\mathcal{O}}$ and $[t+1, T]$, we only have to reassign the operation that starts at date t in the previous matching attached to $\mathrm{LB}(t)$, which requires the search for only one augmenting path in $O(nT)$ time.

3.2.2. Formulation of LB^-

This second lower bound, that will be denoted by LB^-, can be computed much faster than LB^+. Even if it is weaker, it requires the computation of only one assignment, which means that its time complexity is equivalent to the computation of the lower bound for the problem without precedence constraints. The key idea of this lower bound is the modification of the assignment costs of the unary operations of the non-sequenced tasks in order to take into account the presence of the sequenced tasks. More precisely, the assignment cost of any unary operation is increased by $c_t^\Sigma = \Sigma(t) - \Sigma(t+1)$. If c_{it} is the assignment cost of an operation from task i at time t, as described in Section 2.1, we now consider the problem in which the assignment cost is $c_{it} + c_t^\Sigma$.

From the definition of LB^+, $\mathrm{LB}^+ = \mathrm{LB}(t_0) + \Sigma(t_0)$ for some $t_0 \leq T$. But:

$$\mathrm{LB}(t_0) + \Sigma(t_0) = \mathrm{LB}(t_0) + \sum_{t=t_0}^{T} c_t^\Sigma + \Sigma(T+1)$$

The last term, $\Sigma(T+1)$, is the minimal cost for scheduling the chain of tasks $1 \rightarrow \cdots \rightarrow k$. Let $\mathcal{M}(t_0)$ be the optimal assignment between the unary operations of tasks $k+1, \ldots, n$ and time slots $t_0, \ldots, T$. We then have:

$$\begin{aligned} \mathrm{LB}(t_0) + \sum_{t=t_0}^{T} c_t^\Sigma &= \sum_{(i,\,t)\in\mathcal{M}(t_0)} c_{it} + \sum_{t=t_0}^{T} c_t^\Sigma \\ &\geq \sum_{(i,\,t)\in\mathcal{M}(t_0)} c_{it} + c_t^\Sigma \end{aligned}$$

This last value is of course greater than the minimal cost LB^Σ for assigning the unary operations of the non-sequenced tasks to the time slots $[1, T]$ with assignment costs $c_{it} + c_t^\Sigma$. LB^- is defined as $\mathrm{LB}^\Sigma + \Sigma(T+1)$. We have just shown that $\mathrm{LB}^- \leq \mathrm{LB}^+$, which proves it is a lower bound for our problem.

In our experiences, we have found that LB^- is very often equal to LB^+. So we based our branch-and-bound procedure on an implementation of LB^-.

3.2.3. Initial and terminal subsequences

Both lower bounds LB^+ and LB^- can be adapted to the problem in which the precedence graph has not only an initial sequence of tasks that must be scheduled before all the other tasks but also a terminal sequence of tasks that must be scheduled after all the other tasks. Let $1 \leq k < k' \leq n$. The operations $1, \ldots, k$ (resp. $k', \ldots, n$) are scheduled in this order at the beginning (resp. end) of the schedule. The operations in $[k+1, k'-1]$ must start after the end of k and before the beginning of k'. There are no precedence relations between them.

$\Sigma(t)$ is defined as above and $\Sigma'(t')$ is symmetrically the minimal cost for scheduling the chain $k' \rightarrow \cdots \rightarrow n$ after time t'. We define $\mathrm{LB}(t, t')$ for $t' - t > \sum_{i=k+1}^{k'-1} p_i$ as the lower bound derived from LB for scheduling the operations $k+1, \ldots k'-1$ in the time interval $[t, t']$. LB^+ is then $\mathrm{LB}^+ = \min_{t,\,t'} \Sigma(t) + \mathrm{LB}(t, t') + \Sigma(t')$. LB^- is the cost of the assignment of the operations in $\mathcal{O}_{[k+1,\ k'-1]}$ when the cost of each edge (i, t) is $c_{it} + (\Sigma(t) - \Sigma(t+1)) + (\Sigma'(t) - \Sigma'(t-1))$.

The computation of LB^+ requires $O(T^2)$ computations of $\mathrm{LB}(t, t')$ whereas only one is needed for LB^-.

3.3. Branch-and-bound algorithm

The branch-and-bound procedure presented in this section uses the information obtained by the computation of the lower bound presented in Section 2. From the preemptive schedule corresponding to the assignment of the unary operations to the time slots, a non-preemptive schedule is derived and used both as an upper bound in Section 3.3.1 and as a heuristic to rank the outgoing branches of the choice points. This transformation of a preemptive schedule into a non-preemptive one is based on the concept of α-points introduced by Hall, Shmoys, and Wein (1996) and Phillips, Stein, and Wein (1995).

3.3.1. Initial upper bound

From Section 3.1, building a good feasible schedule for $1\|\sum \alpha_i E_i + \beta_i T_i$ can be done by finding a good sequencing of the tasks. Given a preemptive schedule, such an order can be derived from the α-points C_i^α of each task in the preemptive schedule. For any $\alpha \in (0, 1]$, C_i^α is defined as the earliest point in time when the α-fraction of task i has been completed. By "continuity" in α, C_i^0 is the start time of job i in the preemptive schedule. A feasible schedule can then be scheduled by taking the jobs in the order of their α-points. Different such schedules can be computed for different α-values in order to obtain the best possible initial upper bound. Section 4 shows that this heuristic derives near optimal schedules. We set $\alpha = 0.5$ to compute the initial solutions.

3.3.2. Branching scheme

The branching scheme builds the optimal sequence from the first task to the last one: a node $\mathcal{N}$ at depth k corresponds to a partial sequence of k tasks fixed in the first k positions, this sequence is called an *initial sequence*. This node has $n-k$ descendant nodes, one for each non-sequenced task. The $n-k$ descendant branches are explored in the order of the α-points of the corresponding tasks. These α-points are derived from the preemptive schedule given by the computation of the lower bound for the current node $\mathcal{N}$. Experimentally, for a branching scheme based on the initial sequence, a smaller α seems to be a better choice. In the experiments reported in this paper, α is set to 0, this corresponds to the order of the tasks given by the start times of the preemptive solution.

Note that it would also be possible to have a backward approach, that is, the tasks are scheduled from the last one to the first one. In such an approach, each node corresponds to a so-called *terminal sequence* and the branches are explored in the decreasing order of the α-points. Finally, we can also adopt a mixed approach. At each node, there are both an initial and a terminal sequence. At each step, we can heuristically choose between an initial branching and a terminal branching. If the initial branching is selected, there is one branch per non-sequenced task: this task is added at the end of the initial sequence. Otherwise, there is also one branch per non-sequenced task but the task is added at the beginning of the terminal sequence.

3.3.3. Dominance rules

We now present sufficient conditions for a partial schedule not to be optimal. This partial schedule is here defined by the precedence arcs of graph G_k, which corresponds to a node at depth k. Let τ_k be the completion time of job k in a schedule minimizing the cost of the chain $1 \to \cdots \to k$ and let π_k be the partial sum of the processing times $\pi_k = \sum_{i=1}^{k} p_i$. In any optimal schedule in which the jobs satisfy the precedence relations defined by G_k, it is easy to see that $\pi_k \le C_k \le \tau_k$.

Let us consider an optimal schedule $S_1, \ldots, S_n$ that satisfies the precedence relations of G_k. If we can re-sequence the jobs $1, \ldots, k$ such that we can schedule these k tasks before C_k with a cost less than $\sum_{i=1}^{k} f_i(C_i)$, then we can build a total schedule that is better than $S_1, \ldots, S_n$. As a consequence, if for any possible value of C_k in the interval $[\pi_k, \tau_k]$, we can show that the optimal schedule of the jobs $1, \ldots, k$ such that they all complete before t has a cost strictly less than $\Sigma(t)$, then we are sure that the solution of $1|G_k|\sum \alpha_i E_i + \beta_i T_i$ is not an optimal solution of $1\|\sum \alpha_i E_i + \beta_i T_i$. So in the branch-and-bound algorithm, the node corresponding to the initial sequence $1, \ldots, k$ can be discarded.

For computational reasons, we will limit our search to the case where there is a re-sequencing of $1, \ldots, k$ such that for any possible t, the cost of scheduling this sequence before time t is always strictly less than $\Sigma(t)$. The dominance rule given hereafter is valid when all the cost functions f_i are convex.

Let $S_1^k, \ldots, S_k^k$ be an optimal schedule of the chain of jobs $1 \to \cdots \to k$. Let $\tau_i = S_i^k + p_i$. Let $i, i+1, \ldots, j$ be a subsequence of tasks that are scheduled without idle time in this schedule. We know that when the chain is constrained to complete before a date $t \leq \tau_k$, there always exists an optimal schedule such that the tasks $i, i+1, \ldots, j$ are scheduled without idle time and $C_j \leq \tau_j$. Let B be the block of the jobs $i+1, \ldots, j-1$ (see Figure 2). Let $f_B(S_B)$ be the cost function of scheduling the block B in function of its start time S_B, which is equal to S_{i+1}. Since $f_B(S_B) = \sum_{v=i+1}^{j-1} f_v(S_B + \sum_{\eta=i+1}^{v} p_\eta)$, f_B is also a convex function. Let $f_{iBj}(t)$ (resp. $f_{jBi}(t)$) the cost function of scheduling without idle time at time t, i (resp. j) then B and then j (resp. i). If we can prove that, for any $t \leq S_i^k$, $f_{jBi}(t) < f_{iBj}(t)$ then we know that our initial sequence cannot lead to an optimal schedule. In order to have a sufficient—and easy to compute—condition for the dominance, we now calculate an upper bound for $f_{jBi}(t) - f_{iBj}(t)$. With $p_B = \sum_{\nu \in B} p_\nu$, we have that $f_{jBi}(t) = f_j(t) + f_B(t+p_j) + f_i(t+p_j+p_B)$ and $f_{iBj}(t) = f_i(t) + f_B(t+p_i) + f_j(t+p_i+p_B)$. Therefore, $f_{jBi}(t) - f_{iBj}(t) = (f_i(t+p_j+p_B) - f_i(t)) + (f_B(t+p_j) - f_B(t+p_i)) + (f_j(t) - f_j(t+p_i+p_B)) = \Delta_i(t) + \Delta_B(t) + \Delta_j(t)$ where the three delta function corresponds to the three terms between parenthesis in the previous equation. But we know that for any convex function f and for any $\delta > 0$, $f(t+\delta) - f(t)$ is a nondecreasing function. Therefore, when t is between π_{i-1} and S_i^k, the function in $\Delta_i(t)$, is maximized when $t = S_i^k$, that is:

$$\Delta_i(t) \leq \Delta_i(S_i^k) = f_i(\tau_j - p_i) - f_i(S_i^k)$$

$\Delta_j(t)$ is nonincreasing so that:

$$\Delta_j(t) \leq \Delta_j(\pi_{i-1}) = f_j(\pi_{i-1}) - f_j(\pi_{j-1})$$

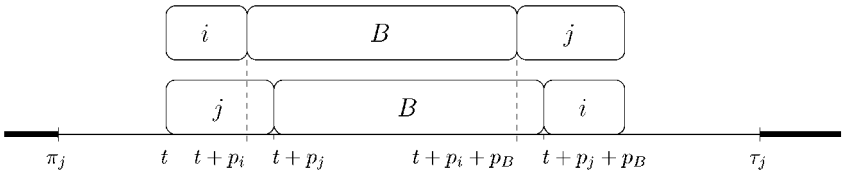

Figure 2. Dominance rule by swapping i and j

The variation of $\Delta_B(t)$ depends on the sign of $p_i - p_j$. It is nondecreasing if $p_j > p_i$ and nonincreasing if $p_i < p_j$.

$$\Delta_B(t) \leq \begin{cases} \Delta_B(S_i^k) & \text{if } p_i < p_j \\ 0 & \text{if } p_i = p_j \\ \Delta_B(\pi_{i-1}) & \text{if } p_i > p_j \end{cases}$$

That is,

$$\Delta_B(t) \leq \begin{cases} f_B(S_i^k + p_j) - f_B(S_{i+1}^k) & \text{if } p_i < p_j \\ 0 & \text{if } p_i = p_j \\ f_B(p_j + \pi_{i-1}) - f_B(\pi_i) & \text{if } p_i > p_j \end{cases}$$

By summing these upper bounds, we obtain the three following sufficient conditions to discard the initial sequence :

(1) $p_i < p_j$ and $f_i(\tau_j - p_i) + f_B(S_i^k + p_j) + f_j(\pi_{i-1} \leq f_i(S_i^k) + f_B(S_{i+1}^k) + f_j(\pi_{j-1})$.
(2) $p_i = p_j$ and $f_i(\tau_j - p_i) + f_j(\pi_{i-1}) \leq f_i(S_i^k) + f_j(\pi_{j-1})$.
(3) $p_i > p_j$ and $f_i(\tau_j - p_i) + f_B(p_j + \pi_{i-1}) + f_j(\pi_{i-1}) \leq (f_i(S_i^k)) + f_B(\pi_i) + f_j(\pi_{j-1})$.

For each pair of tasks (i, j) such that $i < j \leq k$, we can verify in linear time whether one of these dominance criteria is met.

The dominance rule previously described is based on an interchange argument between the jobs i and j: the dominating sequence differs from the initial sequence by a swap between these two jobs. We can also derive a similar dominance rule in which the dominating sequence is derived from the initial sequence by the move of a single job.

Another dominance rule is used in the branch-and-bound algorithm. It is based on the observation that if the partial sequence induces that the remaining jobs will be surely late, then it is possible to derive the optimal schedule that integrates the partial sequence by solving a $1\|\sum \beta_j C_j$ problem for the remaining jobs. The algorithm used to solve it is based on the Smith's ratio rule, that is the jobs are sequenced in order of nondecreasing ratios p_j/β_j (see Smith (1956)).

4. EXPERIMENTAL RESULTS

We randomly generated some instances using an algorithm similar to the one used by Hoogeveen and van de Velde (1996). The branch-and-bound algorithm was tested on instances with 8, 10, 15, 20, 25 and 30 jobs. The processing times are generated from the uniform distribution [10, 100]. The due dates are generated from the uniform distribution [max(0, $P(1 - 3R/2)$), $P(1 + R/2)$], where $P = \sum_{j \in \mathcal{J}} p_j$ and R is a parameter. For each value of R and n, we generated five instances. Each instance was considered with α and β running from 1 to 5. When R is large, the due dates are scattered between 0 and $P(1 + R/2)$ so that the problem is not too difficult in practice: the optimal sequence is nearly the order of the due dates. When $R = 0$, all the tasks have a common due date and a specific algorithm should be used to solve this case. For the parameter R, we considered the values 0.2, 0.4, 0.6, 0.8, and 1.0. According to Hoogeveen and van de Velde (1996), the instances become more difficult with smaller values of R, $R = 0.2$ corresponds to very difficult instances.

We have said in Section 2.3 that it is not a viable branch-and-bound approach to compute at each node the subgradient optimization for $\max_\lambda \mathrm{LB}(\lambda)$ or for $\max_\mu \mathrm{LB}(\mu)$. The best lower bound (for the tradeoff between quality and computation time) we found is $\mathrm{LB}(\lambda^\star)$ where $\lambda^\star$ is the vector such that $c_{it} = 0$ for all the $t \in [d_i - p_i + 1,\ d_i - 1]$ (see Section 1.1). We can easily see that $c_{it}^{\lambda^*} \geq 0$ for any $i \in \mathcal{J}$ and for any $t \in \mathcal{T}$ and that $\lambda^\star$ is the only vector that satisfies this property. For these reasons, $\mathrm{LB}(\lambda^\star)$ is likely to be a good lower bound for the problem.

Experimentally, $\mathrm{LB}(\lambda^\star)$ also seems to be a good lower bound. The gap between this bound and the best known upper bound rendered by the algorithm presented in Section 3.3.1 is about 15% when $n = 10$ and 10% when $n = 20$. In comparison with the lower bound of Rivreau (1999) for the one-machine problem with general cost functions, $\mathrm{LB}(\lambda^\star)$ is weaker but it can be computed faster.

We implemented a branch-and-bound algorithm (coded in C++) with a branching scheme based on the initial sequence, the lower bound $\mathrm{LB}(\lambda^\star)$ and the dominance rules of Section 3.3.3.

Table I exhibits a summary of our computational results on a Pentium III 500 MHz. We present the average number of nodes and the average number of seconds computed over five instances. All averages of integers are rounded up to the nearest integer for the number of nodes.

As expected, the algorithm is very fast, and the results obtained outperform the best ones known for this problem. Unsurprisingly, instances are harder to solve when n is greater or R is smaller. These results show that we can solve very easily small instances of the problem in terms of the number of jobs even if these instances are difficult (small values of R). For $n = 30$, we can see that the algorithm requires much effort, for $R = 0.2$, the number of nodes examined is greater that 100,000 in average. We can notice that the dominance rules are very effective. Computational experiments show that when no dominance rules are included for $n = 15$, the CPU time observed is about a factor of four greater than those shown in Table I. For these instances, the number of nodes examined is 30% greater. These ratios increase substantially for larger values of n. More generally, we can see that the global behavior of the algorithm is good. Computational limits prohibit us from extending the experimental design with more jobs especially for small values of R, but we can expect an improvement of the performance by tuning more appropriate dominance rules in the branch-and-bound algorithm.

Table I. Computational results

n			8					10		
R	0.2	0.4	0.6	0.8	1.0	0.2	0.4	0.6	0.8	1.0
Nodes	114	65	50	36	30	125	214	117	61	108
Seconds	0.01	0.01	0.01	0.01	0.01	0.03	0.04	0.01	0.01	0.01
n			15					20		
R	0.2	0.4	0.6	0.8	1.0	0.2	0.4	0.6	0.8	1.0
Nodes	2064	692	919	820	497	7073	10610	1138	1907	1075
Seconds	0.44	0.14	0.17	0.15	0.09	2.19	2.93	0.27	0.50	0.24
n			25					30		
R	0.2	0.4	0.6	0.8	1.0	0.2	0.4	0.6	0.8	1.0
Nodes	51055	57432	6922	2920	1572	1216708	97640	9021	14398	6040
Seconds	15.49	15.70	1.78	1.07	0.43	403.7	26.34	3.01	4.09	1.69

5. CONCLUSION

We presented a branch-and-bound algorithm to solve the one-machine problem with distinct deadlines and distinct earliness and tardiness penalties. The lower bound is based on the decomposition of the tasks into unary execution time operations so that it can be computed only in pseudo-polynomial time. However, it can be implemented efficiently since in practice the branch-and-bound algorithm explores more that 3000 nodes per second. Problems with 30 tasks can then be solved within minutes so that we plan to extend this approach to solve shop scheduling problems with earliness and tardiness costs.

Further research will be devoted to find an algorithm to compute the lower bound in polynomial time (in the number of jobs) for the earliness–tardiness problem. This lower bound could also be used to derive approximation algorithms with performance guarantee.

APPENDIX A: THE LOWER BOUND AS A LAGRANGEAN RELAXATION

This annex presents an integer program (IP) for the problem with integer start times and general cost functions. A lower bound is derived from a Lagrangean relaxation of this IP. We show how this lower bound is related to LB(λ) defined in Equation (1). This gives a method to maximize LB(λ).

In the IP, each task i is divided in p_i operations that must be started one after the other. The binary variable x_{ijt} is equal to 1 if—idle and non-idle—operation o_{ij} of the task i starts at time t and 0 otherwise. In order to avoid some side effects on the time horizon, and without loss of generality, we can assume that the time slots 1 and T are assigned to an idle operation, namely the operation $o_{0,1}$ and $o_{0,p0}$—see Equation (11).

$$\min_{x} \sum_{(i,j)\in\bar{\mathcal{O}},\, 1<t<T} c_{it}^{0} x_{ijt} \tag{6}$$

$$\sum\nolimits_{(i,j)\in\bar{\mathcal{O}}} x_{ijt} = 1 \quad \forall t \in \mathcal{T} \tag{7}$$

$$\sum\nolimits_{t\in\mathcal{T}} x_{ijt} = 1 \quad \forall (i,j) \in \bar{\mathcal{O}} \tag{8}$$

$$x_{ijt} \in \{0,1\} \quad \forall t \in \mathcal{T}, \forall (i,j) \in \bar{\mathcal{O}} \tag{9}$$

$$x_{ijt} = x_{i(j+1)(t+1)} \quad \forall t \in [1, T-1], \forall (i,j) \in \Omega \tag{10}$$

$$x_{0,1,1} = x_{0,p_0,T} = 1 \tag{11}$$

Equations (6–9) are the usual linear formulation of the assignment problem between the elements of $\bar{\mathcal{O}}$ and the elements of $\mathcal{T}$. Additional Equation (10) say that all the operations of a task $i \in \mathcal{J}$ have to be scheduled without preemption—we recall that $\Omega = \bigcup_{1\in\mathcal{J}} \{(i,\ 1), \ldots, (i,\ p_i - 1)\}$. Indeed, we can assume without loss of generality that the operations $o_{i1}, o_{i2}, \ldots, o_{ipi}$, of task $i \in \mathcal{J}$ are sequenced in this order. Then, if i starts at time t, o_{i1} is assigned to time t and o_{i2} must be assigned to time $t+1$, which is ensured by Equation (10). Similarly, o_{i3} follows o_{i2} and so on. Conversely, if o_{ij} is not assigned to time t, $o_{i(j+1)}$ cannot be assigned to time $t+1$. The idle

task 0 is the only task that can be preempted. So, the integer program renders a solution that corresponds to an optimal schedule for the problem with general cost functions. In order to relax the equalities (10), we introduce a Lagrangean multiplier μ_{ijt} for each inequality and for each vector μ of Lagrangean multipliers, a lower bound denoted $\mathcal{LB}(\mu)$ is obtained by solving the relaxed problem. Note that in this relaxed problem, two UET operations from the same job may have different assignment costs so that this bound must be computed by the $O(T^3)$ Hungarian algorithm. It can be shown that there exists a linear function $\lambda \rightarrow \mu(\lambda)$ such that for any λ, $\mathrm{LB}(\lambda) = \mathcal{LB}(\mu(\lambda))$. More details can be found in Sourd (2000).

As a consequence, $\mathrm{LB}(\lambda)$ and $\mathcal{LB}(\mu(\lambda))$ can be both maximized by the subgradient method. However, experiments have shown that it takes a lot of time to get the subgradient method to converge. Even if this method is able to render an optimal solution when maximizing $\mathcal{LB}(\mu)$ for a 10-jobs instance, it takes more time than a naive enumeration scheme. Research efforts should be devoted to improve the maximization of LB and $\mathcal{LB}$.

REFERENCES

Baker, K. R. and G. Scudder, "Sequencing with earliness and tardiness penalties: A review," *Oper. Res.*, **38**, 22–36 (1990).

Chrétienne, P., Minimizing the earliness and tardiness cost of a sequence of tasks on a single machine, Tech. Report 1999-007, LIP6, (1996).

Chrétienne, P. and F. Sourd, "Scheduling with convex cost functions," *Theoretical Computer Science* (2002), to appear.

Dell'Amico, M. and P. Toth, "Algorithms and codes for dense assignment problems: The state of the art," *Disc. Appl. Math.*, **100**, 17–48 (2000).

Garey, M. R., R. E. Tarjan, and G. T. Wilfong, "One-processor scheduling with symmetric earliness and tardiness penalties," *Math. Oper. Res.*, **13**, 330–348 (1988).

Hall, L. A., D. B. Shmoys, and J. Wein, "Scheduling to minimize average completion time: Off-line and on-line algorithms," *Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms*. pp. 1996, 142–151

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

Khuller, S. and B. Raghavachari, "Advanced combinatorial algorithms," in M. J. Atallah (ed.), *Algorithms and Theory of Computation Handbook*, CRC Press, 1999.

Kuhn, H. W., "The Hungarian method for the assignment problem," *Naval Res. Logist Quat.*, **2**, 83–97 (1955).

Lawler, E. L., "On scheduling problems with deferral costs," *Manage. Sci.*, **11**, 280–288 (1964).

Lawler, E. L., Combinatorial Optimization: Networks and Matroids, Holt Rinehart and Winston, New York, 1976.

Lenstra, J. K., A. H. G. Rinnooy Kan, and P. Brucker, "Complexity of machine scheduling problems," *Ann. Disc. Math.*, **1**, 343–362 (1977).

Nemhauser, G. L., and L. A. Wolsey, Integer and Combinatorial Optimization, Wiley & Sons, 1988.

Phillips, C., C. Stein, and J. Wein, "Scheduling jobs that arrive over time," in *Proceedings of 4th Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science,* Springer-Verlag, 1995.

Rivreau, D. "Problèmes d'ordonnancement disjonctifs: Régles d'élimination et bornes inférieures," Université Technologique de Compiègne, December 1999, in French.

Smith, W. E., "Various optimizers for single-stage production," *Naval Res. Logist. Quat.*, **3**, 59–66 (1956).

Sourd, F., "Contributions à l'étude et à la résolution de problèmes d'ordonnancement disjonctif," Université Pierre et Marie Curie, Paris, 2000.

van den Akker, M., J. A. Hoogeveen, and S. van de Velde, "Combining column generation and lagrangean relaxation," Memorandum COSOR 98-18 Eindhoven University of Technology, Eindhoven, The Netherlands, 1998.

Verma, S., and M. Dessouky, "Single-machine scheduling of unit-time jobs with earliness and tardiness penalties," *Math. Oper. Res.*, **23**, 930–943 (1998).

Author Queries:

1. In proof page 12, first para, last line, please the change made to author name is okay.
2. Please provide publisher details for Nemhauser G. L. and L.A. Wolsey.