INFORMS Journal on Computing
Vol. 21, No. 1, Winter 2009, pp. 167–175
ISSN 1091-9856 | EISSN 1526-5528 | 09 | 2101 | 0167

DOI 10.1287/ijoc.1080.0287

New Exact Algorithms for One-Machine Earliness-Tardiness Scheduling

Francis Sourd
Centre National de la Recherche Scientifique, LIP6 Laboratory, Université Pierre et Marie Curie, 75252 Paris, France, francis.sourd@lip6.fr

In one-machine scheduling, mixed-integer program time-indexed formulations are often used to provide very good lower bounds through Lagrangian relaxations. To get an improved lower bound, we add valid cuts to a time-indexed formulation and show we still have a Lagrangian relaxation that can be solved as a shortest path in a graph. Two branch-and-bound algorithms are then presented for the earliness-tardiness scheduling problem with either common or general due dates. In both cases, our algorithms outperform the previously published approaches.

Key words: time-indexed formulation; Lagrangian relaxation; earliness-tardiness scheduling; common due date; general due dates
History: Accepted by John Hooker, Area Editor for Constraint Programming and Optimization; received May 2007; revised January 2008; accepted May 2008. Published online in *Articles in Advance* September 17, 2008.

1. Introduction

The time-indexed formulation (Dyer and Wolsey 1990, Sousa and Wolsey 1992) is a well-known way to formulate a single-machine scheduling problem as a mixed-integer problem (MIP). It is based on time discretization: time is divided into unary *periods* (or *time slots*), and a binary variable x_{jt} indicates for each job j and each time slot t whether job j completes at t. The main advantage of this integer program is that the linear relaxation is very good. However, as the number of time-points is pseudopolynomial in the size of the input, the resulting MIP is usually very large and is not directly tractable by MIP solvers (such as ILOG CPLEX) for large instances. Kedad-Sidhoum et al. (2008) show that even solving the linear relaxation is not so easy: a column generation approach is necessary for instances with 60 jobs.

Therefore, much research effort was devoted to tackle this formulation, which enables the efficient derivation of good upper and lower bounds. Most approaches are either Lagrangian relaxations or column generation methods. In this paper, we consider the Lagrangian relaxation approach. Two relaxed problems can essentially be derived from this formulation. First, the number of occurrences of the jobs in the schedule can be relaxed. This approach was followed by Péridy et al. (2003) to minimize the number of late jobs and by Avella et al. (2005) to minimize the weighted completion time. The second approach, introduced by Fisher (1976) and Christofides et al. (1987), consists of relaxing the resource constraints.

Remarkably, the time-indexed formulations lead to good lower bounds when the objective function of the scheduling problem is of the form $\sum_j f_j(C_j)$, even if f_j is not nondecreasing (C_j is the completion time of job j). In scheduling terms, such objective functions are called *nonregular*, and they are often used to model that a *too-early delivery* must be penalized. In practice, earliness costs essentially model holding costs and their minimization is the key idea of the just-in-time (JiT) philosophy.

To reflect these ideas in scheduling problems, a due date d_j is given for each job j as well as an earliness penalty α_j and a tardiness penalty β_j. In a feasible schedule, the completion time C_j indicates if j is early ($C_j < d_j$), on time ($C_j = d_j$), or tardy ($C_j > d_j$). The cost of j is computed accordingly; that is, f_j is formally given by $f_j(C_j) = \max(\alpha_j(d_j - C_j), \beta_j(C_j - d_j))$.

In the simple case where there is a single machine, the problem is known to be NP-complete in the strong sense. Various branch-and-bound algorithms have been provided in the literature (Fry et al. 1987, 1996; Kim and Yano 1994; Hoogeveen and van de Velde 1996; Sourd and Kedad-Sidhoum 2003; Sourd 2004). The most recent algorithms proposed by Sourd and Kedad-Sidhoum (2008) and Yau et al. (2008) show that 40-job instances can usually be solved within a few minutes but a significant number of the 50-job instances remain unsolved within a large time limit. An interesting fact is that the improvements in solving the earliness-tardiness problem are due to better lower bounds even if their computation requires significantly more time. It motivates our approach to get an even better lower bound.

Several authors have studied the *common due date* case that is defined by the constraint that $d_j = d$ for each j. This special case leads indeed to some interesting theoretical results that are presented in the survey of Baker and Scudder (1990) to which we want to add two later approximation schemes (Hall and Posner 1991, Kovalyov and Kubiak 1999). To the best of our knowledge, the best exact algorithm for this problem is due to van den Akker et al. (2002). It combines column generation and Lagrangian relaxation and can solve instances with up to 125 jobs (however, the common due date is assumed to be large; that is, $d \geq \sum_j p_j$).

This paper reports three results. First, we present a new lower bound based on the Lagrangian relaxation of a time-indexed formulation reinforced with valid cuts. The idea of these valid cuts are to integrate some dominance rules in the computation of the lower bound while preserving structural properties to have an efficient (pseudopolynomial) algorithm to compute the lower bound.

Then, we use this lower bound in a branch-and-bound algorithm for the earliness-tardiness problem with general due dates. This algorithm relies on a new branching scheme that builds blocks by merging adjacent tasks. Finally, we apply our lower bound to the common due date case and show the relationship of our approach with the one of van den Akker et al. (2002). The common due date is not assumed to be large. In both general and common due date cases, we show that our new algorithms outperform previous ones because we are respectively able to solve all the 50-job instances for the general case and all the instances of the OR-Library (Beasley 1990), which contains problems with up to 1,000 jobs, in the common due date case.

Section 2 presents the reinforced Lagrangian lower bound. In §3, we introduce the branch-and-bound algorithm based on this lower bound and present experimental results for the earliness-tardiness problem with general due dates. Section 4 is devoted to the specialization of the lower bound to the common due date problem and experimental results are also given. Section 5 contains concluding remarks.

2. Reinforced Lagrangian Relaxation

We first formally introduce the scheduling problem and its time-indexed MIP formulation. Then, we will add valid cuts to this formulation and consider a Lagrangian relaxation.

Let $\mathcal{J} = \{1, \ldots, n\}$ be the set of jobs to be scheduled on a single machine. Let $p_j \in \mathbb{N}$ denote the processing time of job j and let $c_{jt} = f_j(t)$ be the *processing cost* of j if it completes at time t. We also assume that we know the scheduling horizon, denoted by T; thus we consider the time periods $1, 2, \ldots, T$. In the earliness-tardiness scheduling problem, we have $f_j(t) = \max(\alpha_j(d_j - C_j), \beta_j(C_j - d_j))$, and we can define T as $\max_j d_j + \sum_j p_j$. For the simplicity of the presentation, we do not introduce release dates, but the approach can be easily adapted to deal with these constraints.

Let x_{jt} be a binary variable equal to one if task j completes at time t and zero otherwise. We also have the variable x_{0t}, which is equal to one if and only if the machine is idle between $t-1$ and t. The time-indexed formulation of our problem is denoted by (TI).

$$\min \quad \sum_{j \in \mathcal{J}} \sum_{t=1}^{T} c_{jt} x_{jt} \tag{1}$$

$$\text{s.t.} \quad \sum_{t=1}^{T} x_{jt} = 1 \quad \forall j \in \mathcal{J}, \tag{2}$$

$$\sum_{j=0}^{n} \sum_{s=t}^{t+p_i-1} x_{js} = 1 \quad \forall t \in [1, T], \tag{3}$$

$$x_{jt} \in \{0, 1\} \quad \forall j \in \mathcal{J}, \forall t \in [r_j, T - p_j]. \tag{4}$$

The objective function (1) indicates that the sum of all the processing costs is to be minimized. Equations (2) ensure that each job is processed once. Inequalities (3), also referred to as *resource constraints*, state that at most one job can be handled at any time. At some time slots, the load of the machine can be null, which means that the machine is idle ($x_{0t} = 1$). Therefore, it will be useful to consider idle time as job 0 whose processing time is equal to one, but job 0 may be processed several times, namely, $T - \sum_j p_j$ times.

We now consider a well-known dominance rule and we code it as a linear constraint that we add to our MIP as a redundant constraint. The role of this redundant constraint is precisely to reinforce the Lagrangian relaxation that will be introduced next.

In a feasible schedule S, let us assume that job i completes at t and job j starts at t (i or j may be equal to zero). As illustrated by Figure 1, the cost of the two adjacent jobs is $c_{it} + c_{j,t+p_j}$. If we swap the pair of jobs (the start times of the other jobs are left unchanged), their cost is then $c_{i,t+p_j-p_i} + c_{j,t+p_j}$. Clearly, if the latter cost is less than the former, S is dominated. Such dominated schedules can be removed

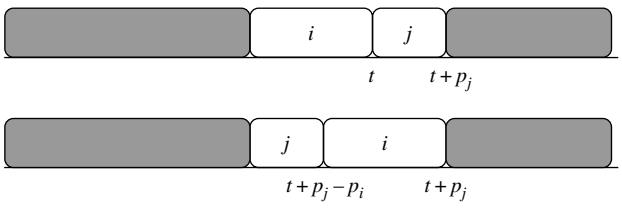

Figure 1 **Swapping Tasks *i* and *j***

from the polyhedron defined by (2) and (3) by adding the linear constraints

$$x_{it} + x_{j,\,t+p_j} \le 1 \quad \text{if } c_{it} + c_{j,\,t+p_j} > c_{j,\,t+p_j-p_i} + c_{i,\,t+p_j}.$$

We can add new valid inequalities by observing that if swapping tasks i and j does not change the cost of the schedule, we can force the task with the smaller index (namely, $\min(i, j)$) to be scheduled before the other task (namely, $\max(i, j)$). A last set of valid inequalities can be derived from the observation that i is different from j. Therefore, we have the set of constraints:

$$x_{it} + x_{j,\,t+p_j} \le 1$$
$$\text{if } \begin{cases} c_{it} + c_{j,\,t+p_j} > c_{j,\,t+p_j-p_i} + c_{i,\,t+p_j}, \\ c_{it} + c_{j,\,t+p_j} = c_{j,\,t+p_j-p_i} + c_{i,\,t+p_j} \text{ and } i > j, \\ c_{it} + c_{j,\,t+p_j} = c_{j,\,t+p_j-p_i} + c_{i,\,t+p_j} \text{ and } i = j \ge 1. \end{cases} \tag{5}$$

The new MIP formulation, denoted by (TI+), is then to minimize (1) subject to constraints (2)–(5). We now relax the constraint set (2) using the Lagrangian relaxation technique. Lagrangian multipliers $(\lambda_1, \ldots, \lambda_n)$ are introduced, and for given values of these multipliers, the Lagrangian problem is to solve

$$\min \left(\sum_{j \in \mathcal{J}} \sum_{t=1}^{T} (c_{jt} - \lambda_j) x_{jt} \right) + \sum_{j \in \mathcal{J}} \lambda_j \tag{6}$$
$$\text{s.t. (3), (4), and (5).}$$

We are going to show that the problem is solved as a shortest path in a directed acyclic graph. In a preliminary step, we introduce graph G depicted in Figure 2. The nodes of G are indexed by the pair (i, t) (for $i = 0, 1, \ldots, n$ and $t = 0, 1, \ldots, T$), and the arcs are defined from (i, t) to $(j, t+p_j)$ (if both nodes exist). Such arcs are indexed by the triple (i, t, j). Intuitively, arc (i, t, j) means that job i completes at t and job j starts at t. There is also a target node $T^\star$ whose predecessors are the nodes (i, T) $(0 \le i \le n)$.

Lemma 1. *There is a one-to-one correspondence between the solutions satisfying* (3) *and* (4) *and the paths from* $(0, 0)$ *to* $T^\star$ *in* G.

Proof. The path traverses all the nodes (i, t) such that $x_{it} = 1$. The correspondence can then be easily checked. □

In scheduling terms, the solutions of (3) and (4) are relaxed schedules in which some tasks may be processed several times or may be left unprocessed. They are referred to as *pseudoschedules* by van den Akker et al. (2000). Figure 2 illustrates that a path in G from $(0, 0)$ to $T^\star$ corresponds to a pseudoschedule.

We now give the cost $c_{jt} - \lambda_j$ to each node (j, t) of G and the null cost to all the "idle" nodes $(0, t)$. The notation G_λ will be used when the reference to the multipliers from which the costs are derived is necessary. With such node costs, the cost of a path corresponds to the objective function (6) up to the constant $\sum_{j \in \mathcal{J}} \lambda_j$. Therefore, the minimum of (6) subject to (3) and (4) is derived from the computation of the minimum-cost path in G_λ. We now show how to find a solution that also satisfies the above dominance conditions.

If a pseudoschedule violates an inequality $x_{it} + x_{j,\,t+p_j} \le 1$ of the set of constraints (5), it means that the corresponding path in G traverses the arc (i, t, j). We then derive the graph $G^\star$ from G by removing all the arcs that correspond to an inequality of (5). To emphasize the difference between G and $G^\star$, we observe that for any arc (i, t, j) of G, G also contains the arcs $(j, t - p_i + p_j, i)$. Both arcs correspond to the execution of jobs i and j in the interval $[t - p_i, t + p_j]$, but arc (i, t, j) means that job i precedes job j while arc $(j, t - p_i + p_j, i)$ means j precedes i. In $G^\star$, one (and only one) of these two arcs is removed, namely, the arc that corresponds to the dominated case.

Clearly, a pseudoschedule that corresponds to a path from $(0, 0)$ to $T^\star$ in $G^\star$ cannot be improved by swapping adjacent jobs. We therefore have the following lemma.

Lemma 2. *There is a one-to-one correspondence between the solutions satisfying* (3)–(5) *and the paths from* $(0, 0)$ *to* $T^\star$ *in* $G^\star$.

There are $O(nT)$ nodes, and because a node has at most n successors, there are $O(n^2T)$ arcs. Therefore, the Lagrangian problem is solved in $O(n^2T)$ time.

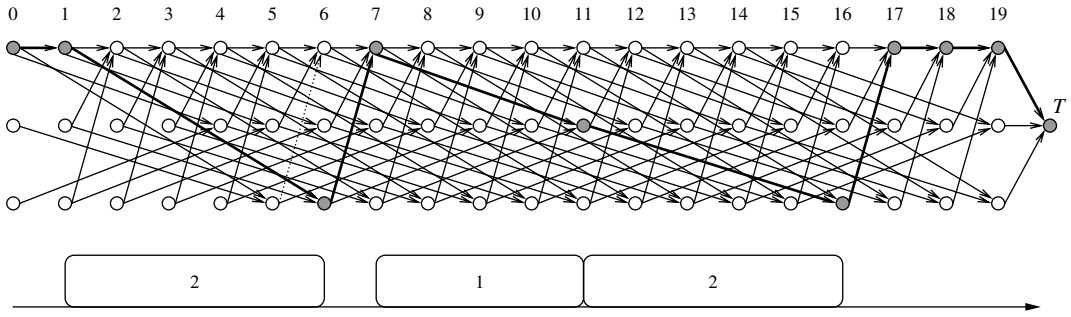

Figure 2 Graph G and a Compatible Pseudoschedule

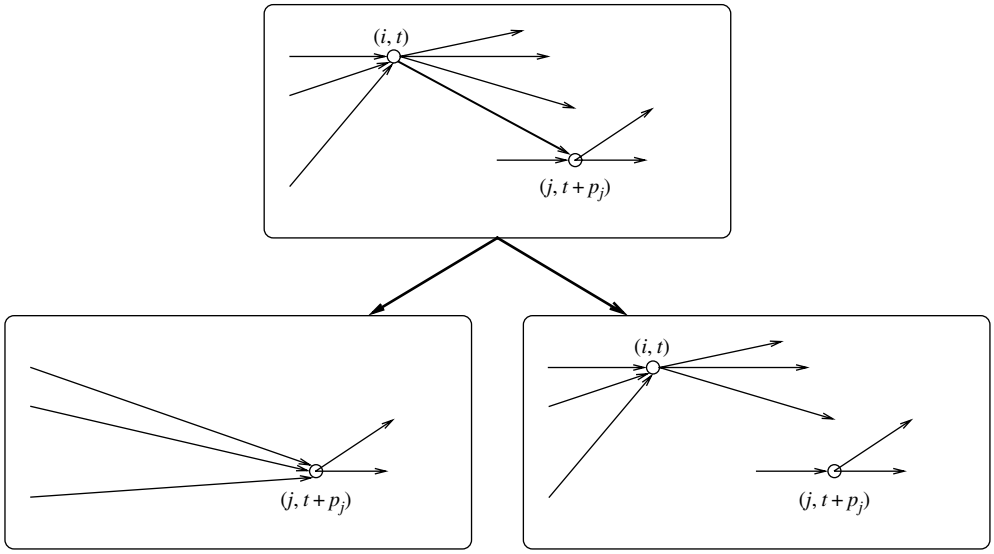

Figure 3 Branching Scheme

Finally, $L(\lambda)$ is maximized with classical nonsmooth convex optimization methods. More details about this point are given in §§3 and 4.

We complete this section by comparing the four lower bounds derived from the linear and Lagrangian relaxations of the two MIP formulations (TI) and (TI+). These lower bounds are, respectively, denoted by lin(TI), lag(TI), lin(TI+), and lag(TI+). The equality between lin(TI) and lag(TI) is well known and due to the integrality property of lag(TI). Due to inequalities (5), lin(TI+) is generally better than lin(TI), and because lag(TI+) does not satisfy the integrality property, lag(TI+) is in general better than lin(TI+). That is,

$$\text{lin(TI)} = \text{lag(TI)} \leq \text{lin(TI+)} \leq \text{lag(TI+)},$$

where the last two inequalities may be strict. From the last case of (5), a pseudoschedule from $G^\star$ is a $P(1)$-path as defined by Péridy et al. (2003), but conversely, some $P(1)$-path may violate the dominance condition defined by (5). Therefore, lag(TI+) is better than the lower bound given by the $P(1)$-paths, but it cannot be compared with the $P(2)$-paths.

3. General Due Dates

In this section, the lower bound is used in a branch-and-bound algorithm for the earliness-tardiness problem with general (distinct) due dates; that is, $f_j(t) = \max(\alpha_j(d_j - C_j), \beta_j(C_j - d_j))$ for each job j. To the best of our knowledge, all the existing branch-and-bound procedures for this problem consist in building a job sequence (remember that when the jobs are sequenced, idle time insertion can be done in polynomial time). Our new lower bound makes it possible to consider a new branching scheme that allows one to take critical decisions earlier in the search tree.

3.1. Branching Scheme

The basic idea of the proposed branching scheme is to consider a pair (i, j) of tasks such that $G^\star$ contains an arc (i, t, j) for some t. As illustrated by Figure 3, the two subproblems generated from the incumbent problem respectively study the schedules in which i and j are adjacent and the schedules in which they are not.

In the first case, we have that j starts when i completes; that is, tasks i and j can be aggregated into a single task—here denoted by k—whose cost function is given by $f_k(t) = f_i(t - p_j) + f_j(t)$. Therefore, the new problem has $n - 1$ tasks. Graph $G^\star$ is modified by merging the nodes (i, t) and $(j, t + p_j)$: for each t, if the arc (i, t, j) exists, then the two nodes are replaced by node $(k, t + p_j)$ (its predecessors are the predecessors of (i, t), and its successors are the successors of $(j, t + p_j)$). If the arc (i, t, j) does not exist, both nodes are removed from $G^\star$.

In the second case, all the arcs between (i, t) and $(j, t + p_j)$ are removed so that any schedule in which job i is immediately followed by job j is forbidden.

When idle time can be inserted between tasks, which is typically the case in the earliness-tardiness scheduling problem with general due dates, this basic idea is not sufficient for the branching scheme to be complete. The problem is that we cannot merge job 0 with job j when we take the decision that there is some idle time before (or similarly after) job j. The reason is that there are generally several idle slots in such a schedule. Instead of merging, we could increase p_j by one (that is, idle time is included in the execution of j) and modify $G^\star$ accordingly. However, if there are t idle slots between two jobs i and j, t such operations may be required before merging jobs i and j in the branching scheme. Because t is not

polynomial in n, it can lead to a search tree with a pseudopolynomial depth.

In our special case of the earliness-tardiness criterion, we can implement a branching strategy that guarantees that the depth of the tree is polynomial in n. When the decision is taken that job j does not start at the completion of another job (that is, $i = 0$ in the selected pair (i, j)), all the arcs entering a node $(j, t + p_j)$ are removed except if the predecessor is $(0, t)$. The case $j = 0$, where job i is set to be followed by idle time, is treated similarly. Therefore, this branching scheme ends when all the tasks (which are in fact blocks of tasks of the root problem) are set to be preceded and followed by idle time. Each block of tasks corresponds to a convex piecewise-linear cost function, and in an optimal schedule, each block must be scheduled at its minimal cost. Therefore, we can check in $O(n)$ time whether this subproblem can lead to an optimal schedule.

The depth of the search tree is clearly in $O(n^2)$. We can modify the branching rule so that the incumbent node has at most n successors: a job j is selected and n branches are created, one for each possible predecessor of j. The depth of the search tree is then in $O(n)$, but each node has at most n descendants instead of two. Our preliminary tests have shown that the former approach is better.

The main issue when implementing this branching scheme is how to select the pair (i, j). Our heuristic is based on the analysis of the pseudoschedule corresponding to the optimum $L(\lambda^\star)$ of L. This pseudoschedule violates some constraints: the main idea is to take decisions to make it feasible. However, $G_{\lambda^\star}$ does not generally contain only one optimal (critical) path, so we will consider the critical graph with respect to the minimum-cost path of $G_{\lambda^\star}$. From this graph, we derive a binary square matrix P of size $n+1$, where P_{ij} is one if and only if there is one critical arc between (i, t) and $(j, t + p_j)$ for some t. Then, for each i, we compute $P^+(i) = |\{j, P_{ij} = 1\}|$ and $P^-(i) = |\{j, P_{ji} = 1\}|$. Finally, the selected pair (i, j) is such that $P_{ij} = 1$ and the sum $P^+(i) + P^-(j)$ is maximum.

3.2. Graph Cleaning

In §2, it is shown that graph $G^\star$ has $O(nT)$ nodes and $O(n^2T)$ arcs, which is quite large for problems with about $n = 50$ tasks. The key of a successful implementation relies on the elimination of a great number of arcs (and nodes) during the branch-and-bound search. We call this procedure *graph cleaning*.

Graph cleaning requires an upper bound, denoted by UB, which corresponds to the best schedule found so far during the search. The idea is to compute a lower bound for the schedules such that i completes at t and j completes at $t + p_j$. If this lower bound is greater than UB, then it means that these schedules cannot lead to an optimal schedule and therefore arc (i, t, j) can be removed from $G^\star$.

For given values of the Lagrangian multipliers $\lambda_1, \ldots, \lambda_n$, we can compute all the lower bounds for all the values i, j, and t in $O(n^2T)$ time. We simply compute the length of the paths from $(0, 0)$ to all the nodes (i, t) and symmetrically the cost of the path from all the nodes (i, t) to the target $T^\star$. Then, we can compute in constant time for each arc a the shortest path from $(0, 0)$ to $T^\star$ that crosses a. Finally, a is removed if the cost of the path is larger than $\mathrm{UB} - \sum_{j=1}^{n} \lambda_j$. Once all the arcs are cleaned out, the nodes (and arcs) that cannot be reached from $(0, 0)$ or that cannot lead to $T^\star$ are also removed.

3.3. Maximizing the Lagrangian Function

In most of the practical uses of the Lagrangian relaxation, the maximization of the Lagrangian function is computed with a subgradient method. In this work, we used a more elaborated algorithm, namely, `Solv0pt`, an implementation of Shor's r-algorithm (Kappel and Kuntsevich 2000). This choice was motivated by the experience of Kedad-Sidhoum et al. (2008), who observe that this algorithm outperforms the subgradient method in the case of the "nonreinforced" Lagrangian relaxation.

Theoretically, graph cleaning should not be called while $L(\lambda)$ is maximized. Indeed, when $G^\star$ is changed, the function L is also changed and so is its maximum. However, in our implementation, it was beneficial to call such graph cleaning while maximizing L. Indeed, we observed that it does not hinder the convergence of the maximization of L. Moreover, because this algorithm is primal, the values returned at each step of the maximization are indeed valid lower bounds and can be used by the branch-and-bound algorithm.

The number of steps to maximize $L(\lambda)$ depends on the initial value of λ. In the branch-and-bound, this initial value is derived from the values of the multipliers computed in the father node. Basically, when jobs i and j are merged, the multiplier of the "merged" tasks is initialized by the half-sum $(\lambda_i + \lambda_j)/2$; the other multipliers are simply initialized with the values of the corresponding multipliers in the father problem.

Because the branch-and-bound only needs a valid lower bound, the maximization of L can stop before the convergence is reached. In our preliminary tests, we tried different strategies, but the best one was to let `Solv0pt` converge.

3.4. Constraint Propagation

When graph cleaning removes some arcs from $G^\star$, it does not only make the computation of $L(\lambda)$ faster, but it also makes the knowledge of the optimal schedules more precise. In particular, by considering for each j the nodes (j, t) that remain in $G^\star$, we are able

to determine time windows in which the jobs must be processed. With these time windows, we can call *edge-finding procedures* that are based on the works of Carlier and Pinson (1989) and that have been widely studied in constraint-based scheduling (Baptiste et al. 2001). Edge-finding derives from the time windows some properties about the order in which the tasks are processed and may thus derive new smaller time windows for the tasks. Once new time windows are computed, nodes (j, t) such that t is no more a possible completion time for j are removed from $G^\star$.

A second constraint propagation algorithm directly works on the arcs of $G^\star$. When job j has only one possible predecessor i in $G^\star$, jobs i and j are immediately merged. A similar algorithm is run when job i has only one successor j.

We have also tested the *shaving technique* that was introduced by Carlier and Pinson (1994) and Martin and Shmoys (1996) for the job-shop scheduling problem. This technique is also widely used as a *forward-checking procedure* in constraint-based scheduling. At any node of the search tree, we check for any pair of jobs (i, j) whether i and j can be adjacent. For each pair, graph $G^\star$ is copied and i and j are merged in this copy graph. Then, the lower bound is computed for the copy graph; if it is greater than UB, then it means the i and j cannot be adjacent and all the arcs (i, t, j) (for all t) are removed from $G^\star$.

3.5. Experimental Results

Two variants of the algorithm (with and without shaving) were implemented in C++, compiled with gcc and run on a Linux Intel Pentium 4 CPU 3.20 GHz. The initial upper bound for the branch-and-bound is computed by the simple local search procedure of Sourd and Kedad-Sidhoum (2008). It is shown that the mean deviation of this heuristic is less than 0.3% for instances with 50 tasks or less.

The tests are based on the benchmark of Sourd and Kedad-Sidhoum (2008) that is available on the Internet at http://www-poleia.lip6.fr/∼sourd/project/et and contains a large number of instances. 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 dates 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. Instances are generated 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), there are 26 instances so that the benchmark contains 5,096 instances.

Figure 4 shows the efficiency of the algorithm (the variant without shaving). There are three curves corresponding to the classes of instances with $n = 30$, 40, or 50 tasks. For a given CPU time t on the log-scale abscissa axis, the curve reports in ordinate the percentage of instances that are solved within time t. This curve is called the *run time distribution*, it is nondecreasing, and reaches 100% when t is large enough. Figure 5 shows similar curves for the branch-and-bound algorithm previously proposed by Sourd and Kedad-Sidhoum (2008).

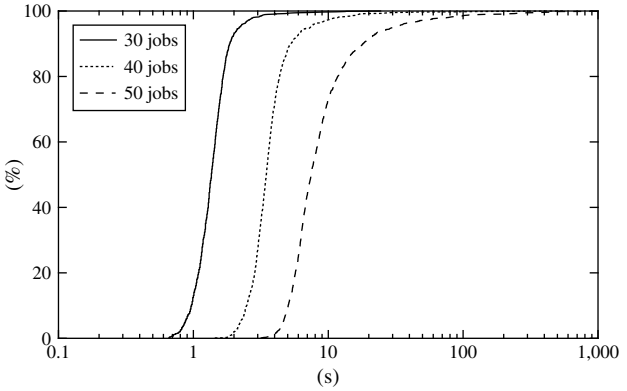

Figure 4 **Percentage of Solved Instances**

In the experimental results of Sourd and Kedad-Sidhoum (2008), only 85% of the 50-job instances were solved within the time limit of 1,000 seconds. Our new algorithm solves all the instances within this time limit, and 85% of the 50-job instances are solved within only 13 seconds. The new algorithm is therefore significantly faster. Moreover, the gradients of the curves in Figure 4 are steeper than the ones in Figure 5. It means than the variance of the CPU times is smaller.

About 64% of the 50 job instances are solved at the root node. When shaving is implemented, about 80% of the instances are solved at the root node but in general, CPU times are significantly larger.

Finally, we tested our algorithm on the benchmark of Bülbül et al. (2007). In this benchmark, there is a wide variety of generation schemes and each task

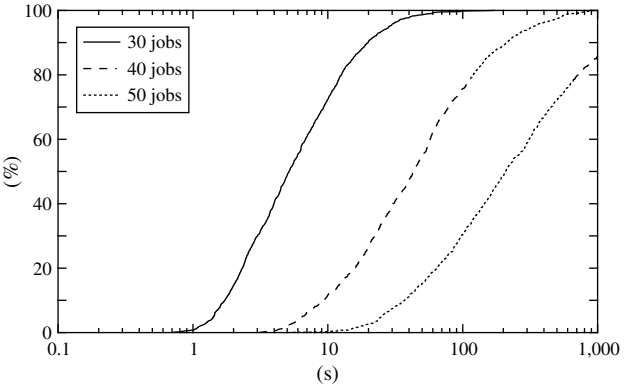

Figure 5 **Percentage of Solved Instances by Sourd and Kedad-Sidhoum (2008)**

has a release date r_j. Our implementation deals with release dates: basically, in $G^\star$, nodes (j, t) for $t < r_j + p_j$ are simply removed. Our algorithm was able to solve all the instances with up to 60 jobs in less than 500 seconds. Comparatively, the previous algorithm (Sourd and Kedad-Sidhoum 2008) leaves some 20-job and 40-job instances open, whereas Bülbül et al. (2007) only present lower and upper bounds.

The main reason why our algorithm performs better is the high quality of the lower bound lag(TI+). However, the efficiency of a branch-and-bound procedure relies on the trade-off between the quality of the bound and its computation time. Here, the computation of $L(\lambda)$ requires $O(n^2T)$ time but graph cleaning significantly improves the practical computation time. Moreover, some preliminary tests have shown that the maximization of $L(\lambda)$ is easier than the maximization of the Lagrangian function derived from lag(TI).

4. The Common Due Date Problem

In the widely studied common due date case, all the jobs have the same due date d, that is, the cost function is given by $f_j(t) = \max(\alpha_j(d-t), \beta_j(t-d))$. According to some strong dominance properties (Baker and Scudder 1990, Hall et al. 1991), we know that there is an optimal schedule in which there is no idle time in between the jobs. Moreover, either there is a job that completes at d or all the jobs are processed in the time interval $[0, P]$ with $P = \sum_{j=1}^{n} p_j$. Finally, the early jobs are scheduled in the order of the nonincreasing p_j/α_j and the jobs that start after d are scheduled in the order of the nondecreasing p_j/β_j. In an optimal schedule that starts at time 0, there may be a so-called *straddling job* that starts before d and completes after d.

In this section, we first show how these well-known properties can be used to improve the complexity of the computation of the shortest path in $G^\star$. Experimental results are then provided.

4.1. Lower Bound Computation

To get a faster lower bound, we first show how to compute the shortest path from $(0, 0)$ to node (j, t) with $t \le d$. We denote by $E(j, t)$ the length of this shortest path. Without loss of generality, we can assume that the tasks are numbered in the nonincreasing order of the p_j/α_j. Therefore, the predecessors of node (j, t) are $\{(0, t-p_j), (1, t-p_j), \ldots, (j-1, t-p_j)\}$. Then, the length of the shortest path to reach (j, t) is equal to $f_j(t) + \tilde{E}(j-1, t-p_j)$, where $\tilde{E}(j, t) = \min_{0 \le i \le j} E(j, t)$. Once all the $E(j, t)$ are computed for a given value of t, all the $\tilde{E}(j, t)$ (for this value of t) can clearly be computed in $O(n)$ time. For some t, if we assume that all the values $\tilde{E}(j, t')$ for $0 \le j \le n$ and $t' < t$ are computed, we can compute the n values $E(j, t)$ (and, subsequently, $\tilde{E}(j, t)$) in $O(n)$ time, which means that all the values $E(j, t)$ for $0 \le j \le n$ and $t \le d$ can be computed in $O(nd)$.

Similarly, we can compute in $O(n(T-d))$ all the values $T(j, t)$ that represent the length of the shortest path from (j, t) to the target $T^\star$. Here, jobs have to be re-indexed according to the p_j/β_j values.

We can finally compute the shortest path from $(0, 0)$ to $T^\star$ by considering all the *straddling arcs*, that is, the arcs (i, t, j) such that $t < d < t + p_j$. Formally, let $S(G^\star)$ be the set of such straddling arcs. The shortest path is given by $\min_{(i,t,j) \in S(G^\star)} E(i, t) + T(j, t + p_j)$. There are at most $n(p_j - 1)$ straddling arcs corresponding to job j because we must have $d - p_j < t < d$. Therefore, the size of $S(G^\star)$ is in $O(nP)$, which means that the shortest path is computed in $O(nT)$. As in §2, this lower bound is denoted by $L(\lambda)$ for a given vector λ of multipliers.

This lower bound has some similarities with the Lagrangian relaxation presented by van den Akker et al. (2002). The main difference is that the latter one works in the large due date case, which means that there is no straddling job. This property is the base of their dynamic programming algorithm that separately builds the pseudoschedule for the early tasks and for the late tasks. Using this lower bound when $d < P$, we have a lower bound for the schedules that have no straddling jobs. This lower bound is denoted by $L^d(\lambda)$. We note that even if d is large, $L(\lambda)$ can be better than $L^d(\lambda)$ because our lower bound takes into account the dominance inequalities (5) between the on-time job and the first late job.

If we force the jobs to be scheduled in the time interval $[0, P]$ without any idle time, that is, we set $T = P$, then $L(\lambda)$ renders a lower bound for a schedule without idle time. This lower bound is denoted by $L^0(\lambda)$. Because a schedule is either in the time interval $[0, P]$ or has no straddling job, we have that $L'(\lambda) = \min(L^d(\lambda), L^0(\lambda))$ is a lower bound for our common due date problem, and so is $\max(L(\lambda), L'(\lambda))$.

4.2. Experimental Results

Our C# implementation uses $L'(\lambda)$, which is generally better than $L(\lambda)$. The maximization of $L'(\lambda)$ is done with a simple subgradient algorithm. An upper bound is derived from the pseudoschedules found when computing the lower bounds. The idea is to make them feasible. Basically, we randomly remove one occurrence of a job that is computed twice (early and tardy) and we insert any missing job after randomly selecting whether it will be early or tardy (if an early insertion is not possible, a tardy insertion is processed).

Tables 1(a)–(d) show the number of solved instances and the average and maximum computation times for the different classes of instances of

Table 1 Solving the Instances of the OR-Library

(a) $h = 0.2$				(b) $h = 0.4$			
n	% solved	Avg. time	Max. time	n	% solved	Avg. time	Max. time
50	100	0.12	0.15	50	100	0.15	0.28
100	100	0.67	0.94	100	100	0.85	0.99
200	100	5.90	7.40	200	100	7.19	8.72
500	100	64.4	78.7	500	100	90.4	105
1,000	100	611	850	1,000	100	794	1,027
(c) $h = 0.6$				(d) $h = 0.8$			
50	100	0.14	0.21	50	100	0.12	0.17
100	100	1.08	1.92	100	100	1.02	1.69
200	100	7.83	10.9	200	100	7.56	10.2
500	100	98.9	139	500	100	101.1	150
1,000	100	915	1,321	1,000	100	918	1,394

the OR-Library (Beasley 1990, Biskup and Feldmann 2001). Tests have been run on a 2 GHz PC laptop. These instances have been widely used to compare different heuristics and metaheuristics.

In all our experiments, the algorithm remarkably converges in the sense that the lower bound found by the subgradient algorithm is always equal to the upper bound given by the heuristic. In other words, no branching has been necessary to prove the optimality of any of these instances. We also underline that the CPU times are quite reasonable: even the largest instances that contain 1,000 jobs can be solved in about 15 minutes.

These results corroborate the experimental conclusions of van den Akker et al. (2002) and extend them to the general common due date problem. However, their approach—which is in fact mainly based on column generation—can solve instances with up to 125 jobs. Our "pure" Lagrangian approach is significantly faster and requires less memory and no linear program solver.

5. Conclusion

This paper has presented a new lower bound for a one-machine scheduling problem and its applications to the earliness-tardiness problems with either a common due date or general due dates. The quality of this lower bound, in spite of long computation times, is the key of the successful solving of larger instances. However, to have an efficient computation of this lower bound, several techniques are used: cleaning and constraint propagation both help to reduce the size of graph G and thus fasten the computation of the shortest path.

Further work should be devoted to adapt this approach to problems with additional constraints. For the problem with *unavailability periods*, Sourd (2007) shows that the resulting constraints can be modeled in the cost function of the jobs. Because the approach works for general cost functions f_j, we can expect a good performance of this algorithm. *Precedence constraints* could be partly taken into account by making a Lagrangian relaxation of these constraints and by removing from G the arcs (i, t, j) when i cannot precede j or when a third job must be processed between i and j. However, for some instances of this problem, the Lagrangian relaxation of the resource constraints may give a better lower bound. Quite similarly, *setup times* can be taken into account when building G but the duality gap is probably larger than in the basic problem.

The parallel machine problem is also of interest because Kedad-Sidhoum et al. (2008) show that the Lagrangian relaxation of the number of job occurrences does not converge well at all. Job-shop scheduling seems even harder to tackle with this approach.

Finally, we will conclude by the fact that the lower bound for $1||\sum \alpha_i E_i + \beta_i T_i$ can still be improved. While this paper was submitted, Tanaka (2007) obtained very promising results for the problem without idle time.

Acknowledgments

F. Sourd's current affiliation is SNCF (Société Nationale des Chemins de fer Français), Paris, France.

References

Avella, P., M. Boccia, B. D'Auria. 2005. Near-optimal solutions of large-scale single-machine scheduling problems. *INFORMS J. Comput.* **17** 183–191.

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

Baptiste, Ph., C. Le Pape, W. Nuijten. 2001. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publishers, Norwell, MA.

Beasley, J. E. 1990. OR-library: Distributing test problems by electronic mail. *J. Oper. Res. Soc.* **41** 1069–1072. http://people.brunel.ac.uk/~mastjjb/jeb/info.html.

Biskup, D., M. Feldmann. 2001. Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates. *Comput. Oper. Res.* **28** 787–801.

Bülbül, K., P. Kaminsky, C. Yano. 2007. Preemption in single machine earliness/tardiness scheduling. *J. Scheduling* **10** 271–292.

Carlier, J., E. Pinson. 1989. An algorithm for solving the job-shop problem. *Management Sci.* **35** 164–176.

Carlier, J., E. Pinson. 1994. Adjustment of heads and tails for the job-shop problem. *Eur. J. Oper. Res.* **78** 146–161.

Christofides, N., R. Alvarez-Valdes, J. M. Tamarit. 1987. Project scheduling with resource constraints: A branch and bound approach. *Eur. J. Oper. Res.* **29** 262–273.

Dyer, M. E., L. A. Wolsey. 1990. Formulating the single machine sequencing problem with release dates as a mixed integer problem. *Discrete Appl. Math.* **26** 255–270.

Fisher, M. L. 1976. A dual algorithm for the one-machine scheduling problem. *Math. Programming* **11** 229–251.

Fry, T. D., G. Leong, T. Rakes. 1987. Single machine scheduling: A comparison of two solution procedures. *Omega* **15** 277–282.

Fry, T. D., R. D. Armstrong, K. Darby-Dowman, P. R. Philipoom. 1996. A branch and bound procedure to minimize mean absolute lateness on a single processor. *Comput. Oper. Res.* **23** 171–182.

Hall, N. G., M. E. Posner. 1991. Earliness-tardiness scheduling problems, I: Weighted deviation of completion times about a common due date. *Oper. Res.* **39** 836–846.

Hall, N. G., W. Kubiak, S. P. Sethi. 1991. Earliness-tardiness scheduling problems, II: Deviation of completion times about a restrictive common due date. *Oper. Res.* **39** 847–856.

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

Kappel, F., A. V. Kuntsevich. 2000. An implementation of Shor's r-algorithm. *Computational Optim. Appl.* **15** 193–205.

Kedad-Sidhoum, S., Y. Rios-Solis, F. Sourd. 2008. Lower bounds for the earliness-tardiness scheduling problem on single and parallel machines. *Eur. J. Oper. Res.* **189** 1305–1316.

Kim, Y. D., C. Yano. 1994. Minimizing mean tardiness and earliness in single-machine scheduling problems with unequal due dates. *Naval Res. Logist.* **41** 913–933.

Kovalyov, M. Y., W. Kubiak. 1999. A fully polynomial approximation scheme for the weighted earliness-tardiness problem. *Oper. Res.* **47** 757–761.

Martin, P. D., D. B. Shmoys. 1996. A new approach to computing optimal schedules for the job shop scheduling problem. S. T. McCormick, W. H. Curnigham, M. Queyranne, eds. *Integer Programming and Combinatorial Optimization*: *Proc. Fifth Internat. IPCO Conf., Vancouver, BC, Canada. Lecture Notes in Computer Science*, Vol. 1084. Springer, Berlin, 389–403.

Péridy, L., E. Pinson, D. Rivreau. 2003. Using short-term memory to minimize the weighted number of late jobs on a single machine. *Eur. J. Oper. Res.* **148** 591–603.

Sourd, F. 2004. The continuous assignment problem and its application to preemptive and non-preemptive scheduling with irregular cost functions. *INFORMS J. Comput.* **16** 198–208.

Sourd, F. 2007. Scheduling with periodic availability constraints and irregular cost functions. *RAIRO Oper. Res.* **41** 141–154.

Sourd, F., S. Kedad-Sidhoum. 2003. The one machine problem with earliness and tardiness penalties. *J. Scheduling* **6** 533–549.

Sourd, F., S. Kedad-Sidhoum. 2008. A faster branch-and-bound algorithm for the earliness-tardiness scheduling problem. *J. Scheduling* **11** 49–58.

Sousa, J. P. De, L. A. Wolsey. 1992. A time-indexed formulation of non-preemptive single-machine scheduling problems. *Math. Programming* **54** 353–367.

Tanaka, S. 2007. An exact algorithm for single-machine scheduling without idle time. P. Baptiste, G. Kendall, A. Munier-Kordon, F. Sourd, eds. *Proc. 3rd Multidisciplinary Internat. Conf. Scheduling*: *Theory and Appl.* (*MISTA*), *Paris*, 314–317.

van den Akker, M., H. Hoogeveen, S. van de Velde. 2002. Combining column generation and Lagrangean relaxation to solve a single-machine common due date problem. *INFORMS J. Comput.* **14** 35–51.

van den Akker, M., C. A. J. Hurkens, M. W. P. Savelsbergh. 2000. Time-indexed formulations for machine scheduling problems: Column generation. *INFORMS J. Comput.* **12** 111–124.

Yau, H., Y. Pan, L. Shi. 2008. New solution approaches to the general single machine earliness-tardiness problem. *IEEE Trans. Automation Sci. Engrg.* **5**(2) 349–360.