Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computers & Operations Research 32 (2005) 2905–2917
computers & operations research
www.elsevier.com/locate/dsw

An exact approach to early/tardy scheduling with release dates

Jorge M.S. Valente*, Rui A.F.S. Alves

Faculdade de Economia, Universidade do Porto, Rua Dr. Roberto Frias, 4200-464 Porto, Portugal

Available online 9 June 2004

Abstract

In this paper, we consider the single machine earliness/tardiness scheduling problem with different release dates and no unforced idle time. The problem is decomposed into weighted earliness and weighted tardiness subproblems. Lower bounding procedures are proposed for each of these subproblems, and the lower bound for the original problem is the sum of the lower bounds for the two subproblems. The lower bounds and several versions of a branch-and-bound algorithm are then tested on a set of randomly generated problems, and instances with up to 30 jobs are solved to optimality. To the best of our knowledge, this is the first exact approach for the early/tardy scheduling problem with release dates and no unforced idle time.

Keywords: Scheduling; Early/tardy; Release dates; Branch-and-bound

1. Introduction

In this paper, we consider a single-machine scheduling problem with early and tardy costs, different release dates and no unforced machine idle time. Scheduling models with both earliness and tardiness costs are compatible with the philosophy of just-in-time production, which emphasizes producing goods only when they are needed, since jobs are scheduled to complete as close as possible to their due dates. It is assumed that no unforced machine idle time is allowed, and therefore the machine is only idle when no jobs are available for processing. This assumption represents a type of production environment where the machine idleness cost is higher than the cost incurred by completing a job early, or the machine is heavily loaded, so it must be kept running in order to satisfy the demand. Korman [1] and Landis [2] give some specific examples of production settings with these characteristics. The existence of different release dates is compatible with the assumption of no unforced idle time, as long as the forced idle time caused by the distinct release dates is inexistent

* Corresponding author. Fax: +351-22-550-50-50.
E-mail address: jvalente@fep.up.pt (J.M.S. Valente).

0305-0548/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2004.04.016

or quite small. If that is not the case, the assumption becomes unrealistic, since the machine capacity is then clearly not limited when compared with the demand and it is unlikely that the cost of the machine being kept idle is higher than the early cost.

Formally, the problem considered in this paper can be stated as follows. A set of n independent jobs $\{J_1, J_2, \ldots, J_n\}$ has to be scheduled without preemptions on a single machine that can handle only one job at a time. The machine is assumed to be continuously available from time zero onwards. Job J_j, $j = 1, 2, \ldots, n$, becomes available for processing at its release date r_j, requires a processing time p_j and should ideally be completed on its due date d_j. Given a schedule, the earliness of J_j is defined as $E_j = \max\{0, d_j - C_j\}$, while the tardiness of J_j can be defined as $T_j = \max\{0, C_j - d_j\}$, where C_j is the completion time of J_j. The objective is then to find a schedule that minimises the sum of weighted earliness and weighted tardiness $\sum_{j=1}^{n}(h_j E_j + w_j T_j)$ subject to the constraint that no unforced machine idle time is allowed, where h_j and w_j are the earliness and tardiness penalties of job J_j.

The problem is strongly NP-hard, since it is a generalization of weighted tardiness scheduling [3]. To the best of our knowledge, the only work on this problem is due to Valente and Alves [4]. They analyse the performance of several dispatch rules, a greedy procedure and a decision theory local search heuristic. The early/tardy problem with equal release dates and no idle time, however, has been considered by several authors. Abdul-Razaq and Potts [5], Li [6] and Liaw [7] developed branch-and-bound algorithms. Ow and Morton [8] proposed several dispatch rules and a filtered beam search procedure. An additional dispatch rule and a greedy algorithm were presented by Valente and Alves [9]. A neighbourhood search heuristic was also presented by Li [6]. The weighted tardiness problem with release dates has also been considered by Akturk and Ozdemir [10,11]. In [10] they develop a dominance rule that is used to improve the performance of several heuristics, while in [11] they present some new dominance rules and lower bounding procedures that are incorporated in a branch-and-bound algorithm.

In this paper, we present a branch-and-bound algorithm based on a decomposition of the problem into weighted earliness and weighted tardiness subproblems. We propose lower bound procedures for each of these subproblems, and the lower bound for the original problem is then the sum of the lower bounds for the two subproblems. We also propose using dominance rules originally derived for the problem with equal release dates in order to eliminate dominated nodes from the search tree. Several versions of the branch-and-bound algorithm are then tested on a set of randomly generated problems with up to 30 jobs.

This paper is organized as follows. In Section 2 we describe the decomposition of the problem and the derivation of the lower bound procedures. The dominance rules are presented in Section 3. Section 4 describes the implementation details of the branch-and-bound algorithm. The computational results are presented in Section 5. Finally, some concluding remarks are given in Section 6.

2. Decomposition of the problem and derivation of the lower bounds

In this section, we first formulate the problem and decompose it into two subproblems with a simpler structure. This decomposition is similar to the one presented by Li [6] for the early/tardy problem with equal release dates. We then present two general lower bound procedures for each of the subproblems. Finally, we describe the specific procedures used to obtain the lower bound.

2.1. Decomposition of the problem

The early/tardy scheduling problem (P) can be formulated as

$$V = \min \sum_{j=1}^{n} (h_j E_j + w_j T_j) \tag{P}$$

s.t.

$$E_j \geqslant 0, \quad j = 1, \ldots, n, \tag{1}$$

$$E_j \geqslant d_j - C_j, \quad j = 1, \ldots, n, \tag{2}$$

$$T_j \geqslant 0, \quad j = 1, \ldots, n, \tag{3}$$

$$T_j \geqslant C_j - d_j, \quad j = 1, \ldots, n, \tag{4}$$

$$r_j \leqslant C_j - p_j, \quad j = 1, \ldots, n, \tag{5}$$

$$\text{machine capacity constraints}, \tag{6}$$

where constraints (1)–(4) reflect the definitions of job earliness and tardiness, and constraint (5) specifies that no job can start before its release date. If we consider only the earliness costs or the tardiness costs in the objective function, it is possible to decompose problem (P) into two subproblems (P_1) and (P_2) with objective functions $V_1 = \min \sum_{j=1}^{n} h_j E_j$ and $V_2 = \min \sum_{j=1}^{n} w_j T_j$, respectively. Constraints (1) and (2) are only relevant to subproblem (P_1), while constraints (3) and (4) are needed only in (P_2).

The motivation for this decomposition is twofold. On the one hand, subproblems (P_1) and (P_2) have a simpler structure than the original problem. On the other hand, subproblem (P_2) is the weighted tardiness problem with release dates, for which a lower bounding procedure already exists. Given that unforced idle time is not allowed, subproblem (P_1) is symmetrical in structure to (P_2), so lower bounding procedures similar to those for (P_2) may be used. Nevertheless, (P_2) is NP-hard, since it is a generalization of weighted tardiness scheduling with equal release dates. Therefore, subproblem (P_1) can also be considered as NP-hard, given its symmetry to (P_2). Even if this were not the case, solving (P_1) and (P_2) would not yield a direct solution to (P). So instead of directly solving the two subproblems, we will develop efficient lower bounding procedures for (P_1) and (P_2) in order to obtain a lower bound for (P).

Theorem 1. *$V_1^* + V_2^* \leqslant V^*$, where V_1^*, V_2^* and V^* are the minimum objective function values of (P_1), (P_2) and (P), respectively.*

Proof. Similar to the proof of Theorem 3.1 in [6]. □

Theorem 2. *If L_1 and L_2 are lower bounds for problems (P_1) and (P_2), respectively, then $L_1 + L_2$ is a lower bound for problem (P).*

Proof. Similar to the proof of Lemma 3.1 in [6]. □

2.2. Lower bound procedures for subproblem (P_1)

We will now present two lower bounding procedures for subproblem (P_1). The first procedure relaxes the assumption that a job cannot be scheduled before its release date and calculates a lower bound for a problem with equal release dates. The second procedure uses a lower bound for the weighted completion time problem with release dates.

Let S be a partial schedule (possibly empty) for problem (P_1) and U be the set of yet unscheduled jobs. Our objective is to obtain a lower bound on the minimum cost of scheduling the jobs in U after the partial schedule S. Let $C^U_{\max}$ be the time at which the last job in U to be scheduled will be completed (since no unforced idle time is allowed, this time is sequence-independent) and $(V_1)^*$ denote the optimal objective function value of problem (P_1) on set U. Finally, let $s^1_U = C^U_{\max} - \sum_{j\in U} p_j$ and let $s^2_U = \max(C^S_{\max}, r^U_{\min})$ denote the time at which the next job to be scheduled will start, where $C^S_{\max}$ is the completion time of the last job in S ($C^S_{\max} = 0$ if $S = \emptyset$) and $r^U_{\min} = \min\{r_j : J_j \in U\}$. The following propositions provide two lower bounds for subproblem (P_1).

Proposition 3. *Given a problem (P_1) on the set of unscheduled jobs U, let (P'_1) be a new problem in which the release dates of all jobs in U are set equal to s^1_U. The following relation holds: $lb^e_1 \leqslant (V'_1)^* \leqslant (V_1)^*$, where $(V'_1)^*$ is the optimal objective function value of (P'_1) and lb^e_1 is any lower bound for that problem.*

Proof. Any permutation of the jobs in U that is feasible for (P_1) is also feasible for (P'_1). Also, the completion time of any job in such a permutation cannot be lower in (P'_1) than it is in (P_1). Therefore, the weighted earliness of each job cannot then be higher in (P'_1) than it is in (P_1), and thus $(V'_1)^* \leqslant (V_1)^*$. □

Proposition 4. *Given a problem (P_1) on the set of unscheduled jobs U, let $lb(\sum[(-h_j)C_j])$ be a lower bound for the weighted completion time problem with release dates $1|r_j|\sum[(-h_j)C_j]$ on set U and starting at time s^2_U. The following relation holds: $lb^e_2 \leqslant (V_1)^*$, where $lb^e_2 = \max(\sum h_j d_j - (-lb(\sum[(-h_j)C_j])), 0)$.*

Proof. Clearly, $(V_1)^* \geqslant 0$ and $h_j d_j - h_j C_j \leqslant h_j E_j$. Let $(\sum h_j C_j)^*$ and $(\sum(-h_j)C_j)^*$ denote the optimum objective function values of the problems $1|r_j|\max\sum h_j C_j$ and $1|r_j|\sum(-h_j)C_j$, respectively. We then have $(V_1)^* \geqslant \sum h_j d_j - (\sum h_j C_j)^*$ and

$$\sum h_j d_j - \left(\sum h_j C_j\right)^* = \sum h_j d_j - \left[-\left(\sum(-h_j)C_j\right)^*\right]$$
$$\geqslant \sum h_j d_j - \left(-lb\left(\sum(-h_j)C_j\right)\right)$$

which concludes the proof. □

2.3. Lower bound procedures for subproblem (P_2)

For subproblem (P_2) we use two lower bounding procedures that were proposed in [11]. The first procedure relaxes the assumption that a job cannot be scheduled before its release date and

calculates a lower bound for the problem with equal release dates, while the second uses a lower bound for the weighted completion time problem with release dates.

Let S be a partial schedule (possibly empty) for problem (P_2) and U be the set of yet unscheduled jobs. Our objective is to obtain a lower bound on the minimum cost of scheduling the jobs in U after the partial schedule S. Let $s_U = \max(C^S_{\max}, r^U_{\min})$ denote the time at which the next job to be scheduled will start, where $C^S_{\max}$ is the completion time of the last job in S ($C^S_{\max} = 0$ if $S = \emptyset$) and $r^U_{\min} = \min\{r_j : J_j \in U\}$. Also let $(V_2)^*$ denote the optimal objective function value of problem (P_2) on set U. The following propositions provide two lower bounds for (P_2).

Proposition 5 (Akturk and Ozdemir [11]). *Given a problem (P_2) on the set of unscheduled jobs U, let (P'_2) be a new problem in which the release dates of all jobs in U are set equal to s_U. The following relation holds: $lb^t_1 \leqslant (V'_2)^* \leqslant (V_2)^*$, where $(V'_2)^*$ is the optimal objective function value of (P'_2) and lb^t_1 is any lower bound for that problem.*

Proposition 6 (Akturk and Ozdemir [11]). *Given a problem (P_2) on the set of unscheduled jobs U, let $lb(\sum w_jC_j)$ be a lower bound for the weighted completion time problem with release dates $1|r_j|\sum w_jC_j$ on set U and starting at time s_U. The following relation holds: $lb^t_2 \leqslant (V_2)^*$, where $lb^t_2 = \max(lb(\sum w_jC_j) - \sum w_jd_j, 0)$.*

2.4. Lower bound procedures for problem (P)

The lower bound methods presented in the previous two subsections are general procedures. Lower bounds lb^e_1 and lb^t_1 can use any lower bound for the weighted earliness and weighted tardiness problems, respectively, while lower bounds lb^e_2 and lb^t_2 can use any lower bound for the weighted completion time problem with release dates. The lower bounds presented by Li [6] were used to calculate lb^e_1 and lb^t_1. Hariri and Potts [12] and Belouadah et al. [13] presented lower bounding procedures for the weighted completion time problem with release dates. We chose the latter lower bound, since preliminary tests indicated its computation time was lower and it provided better or equal results for nearly all of our test instances. The preliminary tests also indicated that the lower bounds for the problem with identical release dates usually provided better results than their weighted completion time problem counterparts. Based on these results, we decided to test four lower bounding procedures, denoted as $E2T2$, $E1T1$, $E2T1$ and $E1T2$. These lower bounds are calculated as follows: $E2T2 = \max(lb^e_1, lb^e_2) + \max(lb^t_1, lb^t_2)$; $E1T1 = lb^e_1 + lb^t_1$; $E2T1 = \max(lb^e_1, lb^e_2) + lb^t_1$ and $E1T2 = lb^e_1 + \max(lb^t_1, lb^t_2)$.

The following theorem shows that, under certain conditions, lower bounds lb^e_1 and lb^t_1 dominate lb^e_2 and lb^t_2, respectively.

Theorem 7. *Let t be the current time and $r_{\max}$ be the largest release date. If $t \geqslant r_{\max}$, we have $lb^e_1 \geqslant lb^e_2$ and $lb^t_1 \geqslant lb^t_2$.*

Proof. If $t \geqslant r_{\max}$, all unscheduled jobs are already available, and lb^t_2 is then equal to $\sum w_jC^{WSPT}_j - \sum w_jd_j$, where C^{WSPT}_j is the completion time of job j when the jobs are scheduled in weighted shortest processing time order (see [13] for details concerning lb^t_2). Lower bound lb^t_1 (see [6] for

details) is obtained by solving the following problem:

$$\max \sum_{j=1}^{n} \lambda_j (C_j^{WSPT} - d_j)$$

s.t.

$$\frac{\lambda_j}{p_j} \geqslant \frac{\lambda_{j+1}}{p_{j+1}}, \quad j = 1, \ldots, n,$$

$$0 \leqslant \lambda_j \leqslant w_j, \quad j = 1, \ldots, n.$$

Since w_j is a feasible value for λ_j, lower bound lb_1^t dominates lb_2^t. A similar reasoning can be used to show that $lb_1^e \geqslant lb_2^e$. □

Based on this result, when the current time is greater than or equal to the largest release date, only the lb_1^e and lb_1^t lower bounds are calculated, and therefore all the procedures become identical to the $E1T1$ lower bound.

3. Dominance rules

In this section, we present the dominance rules that were used to reduce the number of nodes in the search tree. These rules were developed for the problem with identical release dates, but can still be used when the release dates are allowed to be different, provided care is taken to avoid making unfeasible job swaps. Ow and Morton [8] proved that in an optimal schedule all adjacent pairs of jobs J_i and J_j, with J_i preceding J_j, must satisfy the following condition:

$$w_i p_j - \Omega_{ij}(w_i + h_i) \geqslant w_j p_i - \Omega_{ji}(w_j + h_j)$$

with Ω_{xy} defined as

$$\Omega_{xy} = \begin{cases} 0 & \text{if } s_x \leqslant 0, \\ s_x & \text{if } 0 < s_x < p_y, \\ p_y & \text{otherwise,} \end{cases}$$

where $s_x = d_x - t - p_x$ is the slack of job J_x and t is the sum of the processing times of all jobs preceding J_i.

Liaw [7] demonstrated that all non-adjacent pairs of jobs J_i and J_j, with $p_i = p_j$ and J_i preceding J_j, must satisfy the following condition in an optimal schedule:

$$w_i(p_j + \Delta) - \Lambda_{ij}(w_i + h_i) \geqslant w_j(p_i + \Delta) - \Lambda_{ji}(w_j + h_j),$$

where Δ is the sum of the processing times of all jobs between J_i and J_j and Λ_{xy} is defined as

$$\Lambda_{xy} = \begin{cases} 0 & \text{if } s_x \leqslant 0, \\ s_x & \text{if } 0 < s_x < p_y + \Delta, \\ p_y + \Delta & \text{otherwise,} \end{cases}$$

where s_x and t are defined as before.

When different release dates are allowed, the previous conditions must still be satisfied whenever $r_j \leqslant t$. When this is the case, J_i and J_j can be feasibly swapped, and the above rules must still apply.

4. Implementation of the branch-and-bound algorithm

In this section, we discuss the implementation of the branch-and-bound algorithm. We first calculate an upper bound on the optimum schedule cost using the best of the procedures presented in [4]. The decision theory local search heuristic (denoted as DTS) is first used to generate an initial sequence, and the dominance rules described in the previous section are then applied to improve this sequence (see [4] for details). The upper bound value is updated whenever a feasible schedule with a lower cost is found during the branching process.

We use a forward-sequencing branching rule, where a node at level l of the search tree corresponds to a sequence with l jobs fixed in the first l positions. The depth-first strategy is used to search the tree, and ties are broken by selecting the node with the smallest value of the associated partial schedule cost plus the associated lower bound for the unscheduled jobs. We also use some tests to decide whether a node should be discarded or not. In one version of the branch-and-bound algorithm, three tests are used. In the first test, the adjacent rule of Ow and Morton is applied to the two jobs most recently added to the node's partial schedule. In the second test, Liaw's non-adjacent rule is applied. During the initialization, the algorithm checks if at least two jobs have identical processing times and this second test is skipped when all p_js are different. Finally, if the node is not eliminated by the two previous tests, a lower bound is calculated for that node. If the lower bound plus the cost of the associated partial schedule is larger than or equal to the current upper bound, the node is discarded. The non-adjacent rule is of more limited applicability, since it applies only to jobs with identical processing times, and the different release dates can further limit its use. Therefore, we decided to test another version of the branch-and-bound algorithm that does not use the non-adjacent rule and only applies the other two tests. The branch-and-bound algorithms will be identified by the lower bound used, followed by "+N" when the non-adjacent rule test is applied.

5. Computational results

In this section, we present the results from the computational tests. A set of problems with 15, 20, 25, 30, 40, 50, 75, 100, 250, 500 and 1000 jobs was randomly generated as follows. For each job J_j an integer processing time p_j, an integer earliness penalty h_j and an integer tardiness penalty w_j were generated from one of the two uniform distributions $[1, 10]$ and $[1, 100]$, to create low and high variability, respectively. For each job J_j, an integer release date r_j was generated from the uniform distribution $[0, \alpha \sum_{j=1}^{n} p_j]$, where α was set at 0.25, 0.50 and 0.75. The maximum value of the range of release dates α was chosen so that the forced idle time would be small or inexistent. Preliminary tests showed that $\alpha = 1.00$ would lead to excessive amounts of forced idle time, which would be incompatible with the assumption that no unforced idle time may be inserted in a schedule. Instead of determining due dates directly, we generated slack times between a job's due date and its earliest possible completion time. For each job J_j, an integer due date slack s_j^d was generated

Table 1
Lower bound values

var	*n*	Lower bound						
		*E*2*T*2		*E*1*T*2		*E*2*T*1		*E*1*T*1
		avg	*%imp*	*avg*	*%imp*	*avg*	*%imp*	*avg*
Low	15	351	8.82	348	7.69	326	1.12	323
	25	854	2.78	854	2.78	831	0.00	831
	50	3556	1.03	3544	0.68	3532	0.36	3520
	100	13 438	0.47	13 389	0.10	13 424	0.36	13 375
High	15	22 818	3.00	22 669	2.32	22 304	0.67	22 154
	25	65 793	2.58	65 753	2.51	64 181	0.06	64 141
	50	247 567	0.79	247 104	0.60	246 094	0.19	245 631
	100	973 536	0.05	973 536	0.05	973 057	0.00	973 057

from the uniform distribution $[0, \beta \sum_{j=1}^{n} p_j]$, where the due date slack range β was set at 0.10, 0.25 and 0.50. The due date d_j of J_j was then set equal to $d_j = (r_j + p_j) + s_j^d$. For each combination of instance size n, processing time and penalty variability (*var*), α and β, 20 instances were randomly generated. Therefore, 180 instances were generated for each (*var*, n) combination. All the algorithms were coded in Visual C++ 6.0 and executed on a Pentium IV—1500 Mhz personal computer. The lower bounds were calculated for all test instances, while the branch-and-bound algorithm was used to solve to optimality the instances with up to 30 jobs. Throughout this section, and in order to avoid excessively large tables, we will sometimes present results only for some representative cases.

In Table 1 we present the average value of the lower bounds (*avg*) and the relative improvement (*%imp*) over the $E1T1$ lower bound, calculated as $(LB - E1T1)/E1T1 \times 100$, where LB and $E1T1$ represent the average value of the appropriate lower bound and the $E1T1$ lower bound, respectively. These results are averaged over all the 180 instances that correspond to each combination of instance size and processing time and penalty variability. Procedures $E2T2$ and $E1T2$ provide a noticeable increase in the lower bound value over $E1T1$ for the smaller instances, but that increase is negligible for larger problems. The improvement provided by the weighted completion time lower bound is usually higher for the tardiness subproblem, as can be seen from the results for lower bounds $E1T2$ and $E2T1$. The relative improvement over $E1T1$ decreases with the instance size and with the processing time and penalty variability.

In Table 2 we present the average of the relative deviations from the optimum, calculated as $(O - LB)/O \times 100$, where O and LB represent the optimum objective function value and the lower bound value, respectively. The α and β effect on the relative deviation from the optimum for the $E2T2$ lower bound is given in Table 3. The lower bounds performance is poor, since on average they are 50–60% below the optimum. The performance is better when the processing time and penalty variability is low, and it improves as the instance size increases. The lower bounds performance is adequate when α and β are both at their lowest value, and it deteriorates considerably as α and β increase (the only exception being the $(\alpha = 0.75, \beta = 0.50)$ parameter combination). This result is to be expected, since the early/tardy problem lower bounds, which usually provide better results than

Table 2
Relative deviation from the optimum

var	*n*	Lower bound			
		E2T2	*E1T2*	*E2T1*	*E1T1*
Low	15	55.13	55.60	58.29	58.75
	20	54.37	54.37	56.73	56.73
	25	55.67	55.67	56.97	56.97
	30	52.63	52.82	53.79	53.98
High	15	63.23	63.43	63.99	64.19
	20	61.24	61.81	62.45	63.02
	25	60.62	60.64	61.62	61.64
	30	58.25	58.25	58.92	58.92

Table 3
Lower bound *E2T2* relative deviation from the optimum

var	α	$n = 20$			$n = 30$		
		$\beta = 0.10$	$\beta = 0.25$	$\beta = 0.50$	$\beta = 0.10$	$\beta = 0.25$	$\beta = 0.50$
Low	0.25	11.12	22.69	70.43	10.11	21.20	60.59
	0.50	27.41	53.96	86.38	22.22	55.97	88.96
	0.75	56.36	88.84	72.15	56.32	89.91	68.40
High	0.25	17.16	39.07	72.50	12.55	29.66	62.37
	0.50	30.39	69.93	92.56	33.11	64.13	89.16
	0.75	66.55	87.22	75.78	63.21	91.44	78.65

their weighted completion time problem counterparts, should be more accurate for problems with small release date and due date ranges. The early/tardy lower bound was developed for the problem with identical release dates, so its performance is better when the release dates are only slightly scattered. Also, most jobs will likely be tardy when β is low, and as β increases there will be a greater balance between the number of early and tardy jobs. Previous research on the problem with identical release dates has shown that the early/tardy lower bounds perform better when most jobs are indeed tardy (or early). When the number of tardy and early jobs is similar, the problem is much harder, and the lower bounds become more inaccurate.

In Table 4 we present the average of the upper bound relative deviation from the optimum ($\%dev$), calculated as $(UB - O)/O \times 100$, where UB and O represent the upper bound and the optimum objective function values, respectively. The number of times the upper bound procedure generates an optimal solution ($n^o\ opt$) is also given. The average performance of the upper bound is quite good, since it provides results that are 1–2% above the optimum. The upper bound procedure also generates an optimal schedule for about two-thirds of the 15 and 20 job instances, and one-third of the instances with 25 and 30 jobs.

Table 4
Upper bound performance

n	Low *var*		High *var*	
	$\%dev$	n^o *opt*	$\%dev$	n^o *opt*
15	0.65	139	1.05	137
20	1.41	109	1.04	111
25	1.99	84	2.17	64
30	2.00	50	2.27	49

Table 5
Branch-and-bound runtimes (s)

Algorithm	Low *var*				High *var*			
	$n=15$	$n=20$	$n=25$	$n=30$	$n=15$	$n=20$	$n=25$	$n=30$
$E2T2$+N	0.006	0.038	0.264	8.078	0.005	0.037	0.503	14.521
$E2T2$	0.005	0.040	0.310	12.289	0.005	0.037	0.506	14.646
$E1T2$+N	0.005	0.031	0.215	6.338	0.005	0.030	0.387	11.472
$E1T2$	0.005	0.032	0.249	9.612	0.005	0.029	0.390	11.521
$E2T1$+N	0.005	0.030	0.212	6.182	0.004	0.030	0.381	11.069
$E2T1$	0.004	0.032	0.250	9.349	0.004	0.030	0.384	11.104
$E1T1$+N	0.003	0.023	0.163	4.443	0.003	0.023	0.259	7.953
$E1T1$	0.004	0.024	0.190	6.580	0.004	0.023	0.262	7.917

Table 6
Branch-and-bound runtimes for instances with 30 jobs

var	α	Algorithm					
		$E1T1$+N			$E1T1$		
		β=0.10	$\beta=0.25$	$\beta=0.50$	$\beta=0.10$	$\beta=0.25$	$\beta=0.50$
Low	0.25	0.064	0.162	3.302	0.066	0.174	4.293
	0.50	0.223	0.305	6.358	0.229	0.334	7.916
	0.75	0.696	0.906	27.970	0.923	1.211	44.077
High	0.25	0.045	0.348	3.172	0.043	0.354	3.149
	0.50	0.110	0.259	27.802	0.110	0.255	27.263
	0.75	0.271	3.853	35.719	0.266	3.851	35.963

In Table 5 we give the branch-and-bound average computation times (in seconds). In Table 6 we present the effect of α and β on the branch-and-bound runtimes for the 30 job instances and the $E1T1$+N and $E1T1$ versions. The $E1T1$+N algorithm provides the best results. It can be seen

Table 7
Number of nodes generated and relative importance of the fathoming tests

var	Algorithm	$n = 20$					$n = 30$				
		NG	%*EL*	%*LB*	%*A*	%*NA*	*NG*	%*EL*	%*LB*	%*A*	%*NA*
Low	*E*2*T*2+N	1444	83.56	74.21	25.08	0.71	302 728	89.18	69.11	29.89	1.00
	*E*2*T*2	1538	83.55	75.29	24.71	—	455 507	89.20	70.73	29.27	—
	*E*1*T*2+N	1444	83.56	74.21	25.08	0.71	302 728	89.18	69.11	29.89	1.00
	*E*1*T*2	1538	83.55	75.29	24.71	—	455 508	89.20	70.73	29.27	—
	*E*2*T*1+N	1452	83.57	74.00	25.29	0.71	303 085	89.19	68.94	30.06	1.01
	*E*2*T*1	1547	83.57	75.08	24.92	—	455 933	89.21	70.56	29.44	—
	*E*1*T*1+N	1452	83.57	74.00	25.29	0.71	303 085	89.19	68.94	30.06	1.01
	*E*1*T*1	1547	83.57	75.08	24.92	—	455 933	89.21	70.56	29.44	—
High	*E*2*T*2+N	1479	83.78	73.21	26.72	0.07	552 978	89.11	67.22	32.69	0.10
	*E*2*T*2	1484	83.79	73.32	26.68	—	559 609	89.11	67.36	32.64	—
	*E*1*T*2+N	1479	83.78	73.21	26.72	0.07	552 978	89.11	67.22	32.69	0.10
	*E*1*T*2	1484	83.79	73.32	26.68	—	559 609	89.11	67.36	32.64	—
	*E*2*T*1+N	1483	83.79	73.17	26.76	0.07	553 659	89.12	67.05	32.85	0.10
	*E*2*T*1	1488	83.80	73.28	26.72	—	560 328	89.12	67.20	32.80	—
	*E*1*T*1+N	1483	83.79	73.17	26.76	0.07	553 659	89.12	67.05	32.85	0.10
	*E*1*T*1	1488	83.80	73.28	26.72	—	560 328	89.12	67.20	32.80	—

that the increased accuracy of the lower bounding procedures that use the weighted completion time lower bounds is more than offset by their higher computational requirements. The non-adjacent rule should be used, as it usually leads to lower runtimes, even for instances with high processing time and penalty variability (though the reduction in computation time is much lower in this case). For the larger instances, the computation time is higher when the variability is high. The branch-and-bound runtimes also increase significantly with α and β.

In Table 7 we present the average number of nodes generated by the branch-and-bound algorithm (*NG*), as well as the average percentage of these nodes that were eliminated by the three fathoming tests (%*EL*). We also give some data on the relative importance of these tests, namely the average percentage of nodes eliminated by the lower bound (%*LB*), the adjacent rule (%*A*) and, when appropriate, the non-adjacent rule (%*NA*). In Table 8 we present the α and β effect on the average number of nodes generated and the average percentage of nodes eliminated by the lower bound test for the 30 job instances when the *E*1*T*1+N algorithm is used.

The data in Table 7 once again shows that the improvement provided by the weighted completion time lower bound is higher for the tardiness subproblem. The results given by algorithms that only differ in the use of the weighted completion time bound for the earliness subproblem are in fact nearly identical (although the branch-and-bound tree is not the same, the differences in the results would only become visible with additional decimal places). Only a very small percentage of nodes is eliminated by the non-adjacent rule. This result is most likely somewhat influenced by the order in which the two rules are applied, since the adjacent rule can eliminate nodes that would otherwise be fathomed by the non-adjacent dominance condition. The adjacent dominance rule, however, requires

Table 8
Nodes generated and lower bound test importance for $E1T1+N$ on instances with 30 jobs

var	α	$\beta = 0.10$		$\beta = 0.25$		$\beta = 0.50$	
		NG	*%LB*	*NG*	*%LB*	*NG*	*%LB*
Low	0.25	2012	86.23	5773	78.21	189 675	64.25
	0.50	8237	74.65	13 285	72.01	376 089	67.04
	0.75	34 267	62.57	50 935	60.24	2 047 494	55.22
High	0.25	1391	80.15	13 431	73.51	177 118	61.17
	0.50	4624	73.95	11 970	70.98	1 690 264	62.64
	0.75	14 987	60.77	203 215	63.05	2 865 928	57.26

a much lower computational effort, and it's therefore more efficient to apply it before checking the non-adjacent rule. The proportion of nodes fathomed by the non-adjacent rule decreases with the variability of the processing times and increases with the instance size. This result is to be expected, since it's more likely to find two jobs with the same processing time when the number of jobs is high and the processing time variability is low.

As the instance size and the processing time and penalty variability increase, the percentage of nodes fathomed by the adjacent rule tends to increase, and the effectiveness of the lower bound test correspondingly decreases. For the larger instances, the number of nodes generated is much higher when the variability is high. The number of nodes also tends to increase with α and β. The proportion of nodes eliminated by the lower bound test usually decreases as α and β increase, and the importance of the adjacent rule becomes correspondingly higher, since even when the non-adjacent rule is used, it only has a marginal effect.

The number of nodes generated, and correspondingly the runtimes, increase with the release date and due date ranges, as well as with the processing time and penalty variability. Therefore, the problem becomes much harder to solve with a branch-and-bound algorithm as α and β increase, and instances with a higher variability are also more difficult. As we previously remarked, there is a greater balance between the number of early and tardy jobs as β increases, so the problem should indeed be harder. An increase in the range of release dates, on the other hand, reduces the number of feasible schedules. This reduction is not very substantial, however, since we have restricted the value of α in order to avoid unforced idle time, and is more than compensated by the deterioration in the lower bound performance. As we can see from Tables 7 and 8, the relative importance of the lower bound elimination test decreases as both α and β increase, and is also lower for instances with a high variability.

6. Conclusion

In this paper, we considered the single-machine earliness/tardiness scheduling problem with different release dates and no unforced idle time. We decomposed this problem into weighted earliness and weighted tardiness subproblems, and presented lower bounding procedures for each of these

subproblems. A lower bound for the original problem is then obtained by simply adding the lower bounds for the two subproblems. We also proposed using two dominance rules originally derived for the problem with equal release dates in order to eliminate dominated nodes from the search tree. These rules can still be used in the presence of release dates provided a slight adjustment is made. The lower bounds and several versions of a branch-and-bound algorithm were tested on a set of randomly generated problems. The lower bounds were far below the optimum value, but the branch-and-bound algorithm was still able to find optimal solutions, in a reasonable computation time, for problems with up to 30 jobs. The use of the non-adjacent dominance rule, despite its more limited applicability, is also recommended, since it allowed a decrease in the average computation time, particularly when the processing time and penalty variability is low. To the best of our knowledge, this is the first exact approach for the single machine early/tardy problem with release dates and no unforced idle time.

Acknowledgements

The authors would like to thank two anonymous referees for several valuable comments that were used to improve this paper.

References

[1] Korman K. A pressing matter. Video 1994;46–50.
[2] Landis K. Group technology and cellular manufacturing in the Westvaco Los Angeles VH department. Project report in IOM 581, School of Business, University of Southern California, 1993.
[3] Lenstra JK, Rinnooy Kan AHG, Brucker P. Complexity of machine scheduling problems. Annals of Discrete Mathematics 1977;1:343–62.
[4] Valente JMS, Alves RAFS. Heuristics for the early/tardy scheduling problem with release dates. Working Paper 129, Faculdade de Economia do Porto, Portugal, 2003.
[5] Abdul-Razaq T, Potts CN. Dynamic programming state-space relaxation for single machine scheduling. Journal of the Operational Research Society 1988;39:141–52.
[6] Li G. Single machine earliness and tardiness scheduling. European Journal of Operational Research 1997;96: 546–58.
[7] Liaw CF. A branch-and-bound algorithm for the single machine earliness and tardiness scheduling problem. Computers & Operations Research 1999;26:679–93.
[8] Ow PS, Morton ET. The single machine early/tardy problem. Management Science 1989;35:177–91.
[9] Valente JMS, Alves RAFS. Improved heuristics for the early/tardy scheduling problem with no idle time. Computers & Operations Research, to appear. doi:10.1016/j.cor.2003.08.003.
[10] Akturk MS, Ozdemir D. A new dominance rule to minimize total weighted tardiness with unequal release dates. European Journal of Operational Research 2001;135:394–412.
[11] Akturk MS, Ozdemir D. An exact approach to minimizing total weighted tardiness with release dates. IIE Transactions 2000;32:1091–101.
[12] Hariri AMA, Potts CN. An algorithm for single machine sequencing with release dates to minimize total weighted completion time. Discrete Applied Mathematics 1983;5:99–109.
[13] Belouadah H, Posner ME, Potts CN. Scheduling with release dates on a single machine to minimize total weighted completion time. Discrete Applied Mathematics 1992;36:213–31.