

TECHNICAL NOTE

An exact approach to minimizing total weighted tardiness with release dates

M. SELIM AKTURK* and DENIZ OZDEMIR

Department of Industrial Engineering, Bilkent University, 06533 Bilkent, Ankara, Turkey
E-mail: akturk@bilkent.edu.tr

Received July 1999 and accepted November 1999

The study deals with scheduling a set of independent jobs with unequal release dates to minimize total weighted tardiness on a single machine. We propose new dominance properties that are incorporated in a branch and bound algorithm. The proposed algorithm is tested on a set of randomly generated problems with 10, 15 and 20 jobs. To the best of our knowledge, this is the first exact approach that attempts to solve the $1|r_j|\sum w_j T_j$ problem.

1. Introduction

This study deals with scheduling a set of jobs with unequal release dates, r_j , on a single machine to minimize the total weighted tardiness, $(1|r_j|\sum w_j T_j)$. There are n independent jobs $1, \dots, n$, each of which has an integer processing time p_j , a release date r_j , a due date d_j and a positive weight w_j . Jobs are processed without interruption on a single machine that can handle only one job at a time. The machine may be left idle while there are available jobs in the queue. A tardiness penalty T_j is incurred for each time unit that job j exceeds its due date, i.e., $T_j = \max\{0, (C_j - d_j)\}$, where C_j and T_j are the completion time and the tardiness of job j , respectively. The objective is to find a schedule that minimizes the total weighted tardiness criterion of all jobs given that no job can start processing before its release date.

Rinnooy Kan (1976) shows that the total tardiness problem with unequal release dates, $1|r_j|\sum T_j$ is NP-hard. Lawler (1977) shows that the total weighted tardiness problem, $1||\sum w_j T_j$, is strongly NP-hard, implying that $1|r_j|\sum w_j T_j$ is also strongly NP-hard. Enumerative solution methods have been proposed for both weighted and unweighted cases when all jobs are simultaneously available. Emmons (1969) derives several dominance rules for $1||\sum T_j$. Rinnooy Kan *et al.* (1975) and Rachamadugu (1987) extended these results to $1||\sum w_j T_j$. The Branch and Bound (BB) algorithm of Potts and van Wassenhove (1985) can solve problem

instances with up to 40 jobs. Vairaktarakis and Lee (1995) present a BB algorithm to minimize the total tardiness subject to a minimum number of tardy jobs. Szwarc (1993) proves the existence of a special ordering for the single machine Earliness-Tardiness (E/T) problem with job-independent penalties where the arrangement of two adjacent jobs in an optimal schedule depends on their start time. Recently, Akturk and Yildirim (1998) proposed a new dominance rule and a lower bounding scheme for the $1||\sum w_j T_j$ problem which can be used to reduce the number of alternatives in any exact approach.

All the optimizing approaches discussed above assume that the jobs have equal release dates. To the best of our knowledge, we know of no exact algorithm for the $1|r_j|\sum w_j T_j$ problem. Unequal release dates have been considered for other optimality criteria, by Chu (1992a) and Chand *et al.* (1996) for $1|r_j|\sum F_j$, by Hariri and Potts (1983) and Beloudah *et al.* (1992) for $1|r_j|\sum w_j C_j$, and Potts and van Wassenhove (1988) for $1|r_j|\sum w_j U_j$. Chu (1992b) proves some dominance properties and provides a lower bound for the $1|r_j|\sum T_j$ problem. A BB algorithm is then constructed using the previous results of Chu and Portmann (1992) and problems with up to 30 jobs can be solved for certain problem instances, even though computational requirements for larger problems tend to become prohibitive.

In the following section, we discuss the underlying assumptions and present the proposed dominance rules. Lower bounds for the problem are developed in Section 3. We present a BB algorithm along with a numerical example in Section 4. Computational analysis of the BB

*Corresponding author

algorithm is reported in Section 5, and some concluding remarks are provided in Section 6.

2. Dominance properties

In this section we present new dominance properties to eliminate a number of dominated solutions in any exact algorithm. We show that the arrangement of adjacent jobs in an optimal schedule depends on their start times. For each pair of jobs i and j that are adjacent in an optimal schedule, there can be a critical value t_{ij} such that i precedes j if processing of this pair starts earlier than t_{ij} and j precedes i if processing of this pair starts after t_{ij} . For convenience the jobs are indexed in EDD order such that if $d_i < d_j$, or $d_i = d_j$ then $p_i < p_j$, or if $d_i = d_j$ and $p_i = p_j$ then $w_i > w_j$ or $d_i = d_j$ and $p_i = p_j$ and $w_i = w_j$ then $r_i \leq r_j$ for all i and j such that $i < j$.

To introduce the dominance rule, consider schedules $S_1 = Q_1ijQ_2$ and $S_2 = Q_1jiQ_2$ where Q_1 and Q_2 are two disjoint subsequences of the remaining $n - 2$ jobs. Let t be the completion time of the jobs in Q_1 and jobs i and j are available at t , such that $r_i \leq t$ and $r_j \leq t$.

The interchange function $\Delta_{ij}(t)$ gives the cost of interchanging adjacent jobs i and j whose processing starts at time t , where

$$\Delta_{ij}(t) = f_{ij}(t) - f_{ji}(t),$$

$$f_{ij} = \begin{cases} 0 & \max\{r_i, r_j, t\} \leq d_i - (p_i + p_j), \\ w_i(t + p_i + p_j - d_i) & r_j \leq t \text{ and } \\ & d_i - (p_i + p_j) < t \leq d_i - p_i, \\ w_i(r_j + p_j - t) & d_i - p_i \leq t < r_j, \\ w_i(r_j + p_i + p_j - d_i) & t \leq d_i - p_i \text{ and } t < r_j, \\ w_ip_j & \max\{r_j, d_i - p_i\} \leq t. \end{cases}$$

$\Delta_{ij}(t)$ does not depend on how the jobs are arranged in Q_1 and Q_2 but on the start time t of the pair,

- if $\Delta_{ij}(t) < 0$ then, j should precede i at time t ;
- if $\Delta_{ij}(t) > 0$ then, i should precede j at time t ;
- if $\Delta_{ij}(t) = 0$ then, it is indifferent to whether i or j is scheduled first.

There are five conditions for the computation of f_{ij} . For the first condition, both jobs i and j finish on time, so it is indifferent on whether i or j is scheduled first. In the second condition, job i will become tardy if it is not scheduled first. In the third condition, job j arrives after time t and job i will be tardy if it is scheduled after job j (there is also an idle time on the machine before the beginning of job j). In the fourth condition, if job i is scheduled before job j then it can be finished on time, otherwise it will be tardy. In the last condition, job j arrives before time t , and job i will be tardy even if it is scheduled before job j .

The time dependent dominance properties of the $1||\sum w_j T_j$ problem can be determined by looking at points where the piecewise linear and continuous

functions $f_{ij}(t)$ and $f_{ji}(t)$ intersect. When all possible cases are studied, it can be seen that there are at most seven possible points where functions $f_{ij}(t)$ and $f_{ji}(t)$ intersect. These cases and the following propositions are included here without proof and are described in detail in Akturk and Ozdemir (1998).

$$t_{ij}^1 = [(w_i d_i - w_j d_j)/(w_i - w_j)] - (p_i + p_j), \tag{1}$$

$$t_{ij}^2 = d_j - p_i - p_j(1 - w_i/w_j), \tag{2}$$

$$t_{ij}^3 = d_i - p_j - p_i(1 - w_j/w_i), \tag{3}$$

$$t_{ij}^4 = w_j/w_i(r_i + p_i + p_j - d_j) - (p_i + p_j - d_i), \tag{4}$$

$$t_{ij}^5 = [(w_j - w_i)p_i + w_j r_i + w_i(d_i - p_j)]/(w_i + w_j), \tag{5}$$

$$t_{ij}^6 = w_i/w_j(r_j + p_i + p_j - d_i) - (p_i + p_j - d_j), \tag{6}$$

$$t_{ij}^7 = [(w_i - w_j)p_j + w_i r_j + w_j(d_j - p_i)]/(w_i + w_j). \tag{7}$$

As a result, we can state a general rule that provides a sufficient condition for schedules that cannot be improved by adjacent job interchanges. We show that if any sequence violates the proposed dominance rule, then switching these jobs either lowers the total weighted tardiness or leaves it unchanged as stated below in Proposition 1. In this rule, there are two possibilities for each pair of jobs. Either there is at least one breakpoint or an unconditional ordering. A *breakpoint* is a critical start time for each pair of adjacent jobs after which the ordering changes direction such that if $t \leq \text{breakpoint}$, i precedes j , denoted by $i \prec j$, (or j precedes i) and then j precedes i , denoted by $j \prec i$, (or i precedes j). If i unconditionally precedes j , denoted by $i \rightarrow j$, then the ordering does not change, i.e., i always precedes j when they are adjacent, but this does not imply that an optimal sequence exists in which i precedes j .

Before defining the new dominance properties, we will present some definitions. Let J be the set of all jobs to be scheduled, $S(t)$ the set of jobs scheduled before time t , $A(t)$ the set of available unscheduled jobs at time t , i.e., $A(t) = \{i|r_i \leq t\} - S(t)$, $B(t)$ the set of unavailable and unscheduled jobs at time t , i.e., $B(t) = \{k|r_k > t\}$, and $U(t)$ the set of unscheduled jobs at time t , i.e., $U(t) = (A(t) \cup B(t))$.

Proposition 1. *Let job k be the last scheduled job in the sequence at time t given that processing of job k starts at time $t - p_k$. For all unscheduled jobs $i \in U(t)$, if scheduling job i at time t violates the proposed dominance rule, i.e., $i \prec k$ at time $t - p_k$, then scheduling job i right after job k at time t will not lead to an optimal schedule.*

It is well-known that the Shortest Weighted Processing Time (SWPT) rule gives an optimal sequence for the $1||\sum w_j T_j$ problem when either all due dates are zero or all jobs are tardy, i.e., $t > \max_{i \in J} \{d_i - p_i\}$. Under this situation the problem reduces to the total weighted

completion time problem which is known to be solved optimally by the SWPT rule as shown by Smith (1956), in which jobs are sequenced in nonincreasing order of w_i/p_i . For the $1|r_j|\sum w_jT_j$ problem, if scheduling even the job with shortest processing time at time t makes all unscheduled jobs tardy, then the problem reduces to the $1|r_j|\sum w_jC_j$ problem for the remaining unscheduled jobs. Furthermore, let V be the set of pairs (i, j) for which $\Delta_{ij}(t)$ has at least one breakpoint t_{ij} , $i, j \in V$. The largest of these breakpoints is equal to $t_l = \max_{(i,j) \in V} \{t_{ij}^1, t_{ij}^2, t_{ij}^3\}$. The following proposition can be used to find an optimal sequence for the remaining jobs on hand after time t_l . We know that $t_l \leq \max_{i \in J} \{d_i - p_i\}$, so we enlarge the region for which the $1|r_j|\sum w_jT_j$ problem can be solved optimally by the SWPT rule.

Proposition 2. *If either $\min\{t + \min_{j \in A(t)}\{p_j\}, \min_{k \in B(t)}\{r_k + p_k\}\} \geq \max_{i \in U(t)}\{d_i\}$ or $t > t_l$ then the SWPT rule gives an optimal sequence for the remaining unscheduled jobs.*

Proof. $\min\{t + \min_{j \in A(t)}\{p_j\}, \min_{k \in B(t)}\{r_k + p_k\}\}$ is the earliest completion time of the first job scheduled after time t . When the earliest possible completion time of the first scheduled job exceeds the due dates of all unscheduled jobs, this means that all unscheduled jobs are tardy jobs, hence our problem is equivalent to the $1||\sum w_jC_j$ problem. Furthermore, t_l is the last breakpoint for any pair of jobs i, j on the time scale. For every job pair (i, j) , there is either a breakpoint or unconditional ordering ($i \rightarrow j$). The SWPT rule holds for $i \rightarrow j$. If there is a breakpoint then for $t \geq t_{ij}$ the job having a higher w_j/p_j is scheduled first, so the SWPT again holds. Both jobs should be available before a breakpoint $t_{ij}^k \geq \max\{r_i, r_j\}$ for $k = 1, 2, 3$ so that $t_l = \max\{t_{ij}^1, t_{ij}^2, t_{ij}^3\}$. For $t > t_l$, consider a job i which conflicts with the SWPT rule, then we can have a better schedule by making adjacent job

interchanges which either lowers the total weighted tardiness value or leaves it unchanged. If we do the same thing for all of the remaining jobs, we get the SWPT sequence. ■

Propositions 3 and 4 state the conditions to find a job that should be placed in the first or the last position of the remaining sequence, respectively. Furthermore, these propositions can be implemented iteratively to reduce the number of possibilities at time t .

Proposition 3. *If there is a job $i \in A(t)$ such that $p_i \leq \min_{j \in U(t)}\{p_j\}$, $w_i \geq \max_{j \in U(t)}\{w_j\}$, and $d_i \leq \min_{j \in U(t)}\{d_j\}$ then there is an optimal schedule in which job i will be the first job of the remaining sequence.*

Proposition 4. *For any job $k \in U(t)$ at time t , if $r_k \geq \max_{i \in U(t)}\{r_i\}$, $p_k \geq \max_{i \in U(t)}\{p_i\}$, $d_k \geq \max_{i \in U(t)}\{d_i\}$, and $w_k \leq \min_{i \in U(t)}\{w_i\}$ then job k can be placed at the last position of the remaining sequence.*

If jobs are ‘dense’, that is, all jobs become available after the earliest completion time, then the following proposition specifies the global dominance conditions at time t . Proposition 5 can be proved through a pair-wise interchange argument as depicted in Fig. 1(a and b) where interchanging the positions of jobs k and j in schedule S' , leading to schedule S , improves the $\sum w_jT_j$.

Proposition 5. *If $\max_{i \in U(t)}\{r_i\} \leq \min_{k \in B(t), j \in A(t)}\{r_k + p_k, t + p_j\}$ and j and k are two jobs belonging to $U(t)$ such that $r_j \leq r_k$, $\max\{t, r_j\} + p_j \leq \max\{t, r_k\} + p_k$, $w_j \geq w_k$ and $d_j \leq d_k$ then job j dominates job k at time t .*

We can also generalize one of Emmons’ properties for the static problem such that at any time t , a job with the latest release date will be dominated by the jobs

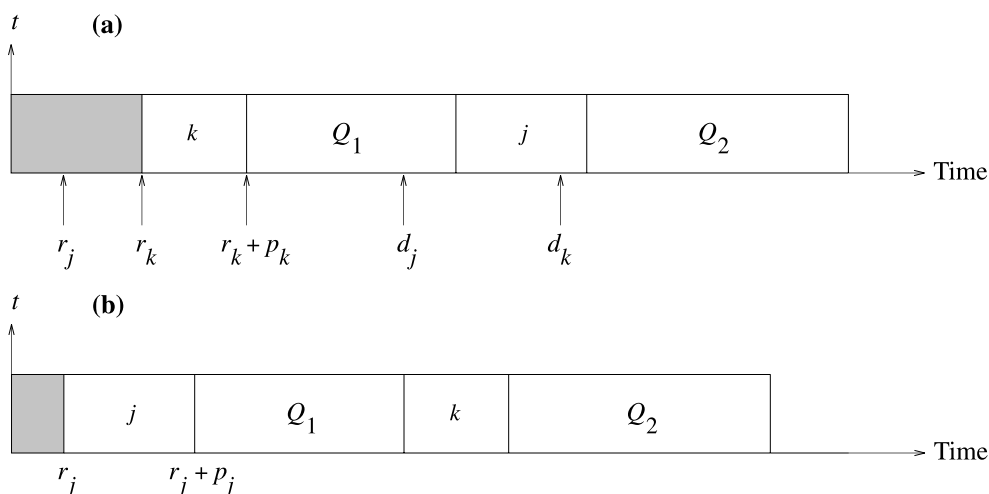


Fig. 1. A possible instance of Proposition 5 for (a) Schedule S' and (b) Schedule S .

with less processing time, earlier due date, and higher weight.

Proposition 6. *If $r_j = \max_{i \in U(t)} \{r_i\}$ then there is an optimal schedule in which job j is preceded by any job $k \in U(t)$ given that $p_j \geq p_k$, $w_j \leq w_k$, and $d_j \geq d_k$.*

At any time t , an unscheduled job with a higher weight, higher processing time and earlier due date dominates another unscheduled job, if its possible Earliest Completion Time (ECT) value is lower than that of the other job. An example of Proposition 7 is given in Fig. 2(a and b) where we exchange the positions of jobs j and i in schedule S' as shown in schedule S .

Proposition 7. *Consider two jobs at time t , such that $i, j \in U(t)$. If $p_i \geq p_j$, $\max\{t, r_i\} + p_i \leq \max\{t, r_j\} + p_j$, $d_i \leq d_j$, and $w_i \geq w_j$ then job i dominates job j at time t .*

If all jobs are available at time t , or a new job arrives after all available jobs are processed, then the set of available jobs forms a block where the dominance rules of Rinnooy Kan *et al.* (1975) and Akturk and Yildirim (1998) apply.

Proposition 8. *If either $\sum_{j \in A(t)} p_j + t \leq \min_{k \in B(t)} \{r_k\}$ or $B(t) = \emptyset$ then unscheduled jobs belonging to $A(t)$ will form a block of jobs such that the following dominance theorem developed by Rinnooy Kan *et al.* (1975) and Akturk and Yildirim (1998) for the $1 || \sum w_j T_j$ problem can be applied.*

- (a) *If one of the following conditions is satisfied job i precedes job j in an optimal sequence, where D_j and C_j be the set of jobs which precede and succeed job j respectively in at least one optimal sequence:*
 - i) $p_i \leq p_j$, $w_i \geq w_j$, and $d_i \leq \max\{d_j, \sum_{h \in D_j} p_h + p_j\}$,

- ii) $w_i \geq w_j$, $d_i \leq d_j$, and $d_j \geq \sum_{h \in S-C_i} p_h - p_j$,
 - iii) $d_j \geq \sum_{h \in S-C_i} p_h$.
- (b) *If $d_i \leq d_j$, $p_i \geq p_j$ and $w_i \leq w_j$, then job j precedes job i for $t \geq t_{ij}$.*

Proof. $B(t) = \emptyset$ means that all unscheduled jobs are available and hence the problem reduces to $1 || \sum w_j T_j$. Otherwise if $\sum_{j \in A(t)} p_j + t < \min_{k \in B(t)} \{r_k\}$ then all available unscheduled jobs will be scheduled before any new job becomes available, then the problem again reduces to the $1 || \sum w_j T_j$ problem for jobs $j \in A(t)$ because the time needed for the arrival of a new job is more than the total processing time of all the available unscheduled jobs. For the $1 || \sum w_j T_j$ problem Rinnooy Kan *et al.* (1975) proved the validity of first three conditions, and the last condition has been proved by Akturk and Yildirim (1998). ■

A feasible schedule is called active if no job can be completed earlier by altering the processing sequence without delaying any other job. An optimal schedule must be an active schedule. The following proposition is needed to guarantee an active schedule. Since scheduling job k at time t violates the definition of an active schedule, it is possible to insert a job before job k without delaying any other job.

Proposition 9. *If $t + \min_{j \in A(t)} \{p_j\} \leq r_k$ then $k \in B(t)$ will not be scheduled at time t in an optimal schedule.*

3. Lower bounding

To the best of our knowledge, there is no lower bound in the literature developed for the $1|r_j| \sum w_j T_j$ problem. A lower bound, which is developed for the $1 || \sum w_j T_j$ problem, can be used as a lower bound for the $1|r_j| \sum w_j T_j$ problem as shown in the following proposition.

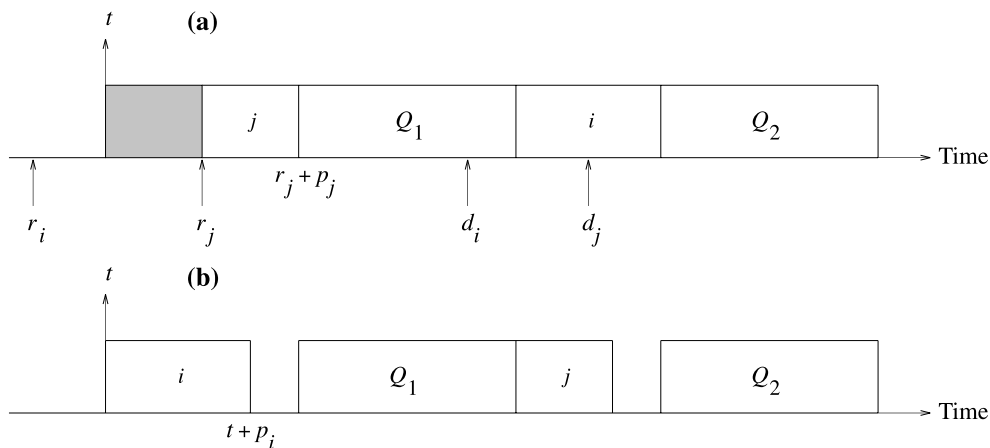


Fig. 2. A possible instance of Proposition 7 for (a) Schedule S' and (b) Schedule S .

Proposition 10. Given a problem Π , construct a new problem Π' in which the jobs are scheduled under the assumption that the jobs are simultaneously available at time zero. The following relation holds: $lb_{\Pi'} \leq (\sum w_j T_j)_{\Pi'}^* \leq (\sum w_j T_j)_{\Pi}^*$, where $(\sum w_j T_j)_{\Pi}^*$ is the optimal total weighted tardiness of problem Π and $lb_{\Pi'}$ is any lower bound obtained for $1 || \sum w_j T_j$.

The linear lower bound for $1 || \sum w_j T_j$ was originally obtained by Potts and van Wassenhove (1985) using Lagrangian relaxation. Abdul-Razaq *et al.* (1990) show that it may also be derived by reducing the objective (total weighted tardiness) to a linear function. This operation is also advantageous in terms of the computational time required to solve the problem. Although it has a weak lower bound value, it can be used in BB algorithms since it can be computed in polynomial time due to low memory requirements (Abdul-Razaq *et al.*, 1990). The linear lower bound takes an initial sequence as its input. Akturk and Yildirim (1998) have shown that the SWPT rule performs well in a reasonable computation time. Therefore, we use the SWPT rule to determine an initial sequence and the corresponding lower bound value, lb_1 , as described in Akturk and Yildirim (1998).

To use the linear lower bound, we relax the assumption that a job cannot be scheduled before its release date. If release dates of the jobs are scattered, the linear lower bound will be very ineffective. Based on a lower bound proposed by Hariri and Potts (1983) for the $1|r_j| \sum w_j C_j$ problem, another lower bound can be derived for the $1|r_j| \sum w_j T_j$ problem by subtracting the total weighted due dates of unscheduled jobs.

Proposition 11. Given a problem Π , suppose \tilde{lb} is any lower bound value for the weighted completion time problem $1|r_j| \sum w_j C_j$, the following relation holds: $\tilde{lb} - \sum w_j d_j \leq (\sum w_j T_j)_{\Pi}^*$, where $(\sum w_j T_j)_{\Pi}^*$ is the optimal total weighted tardiness of problem Π .

The details of how a lower bound can be found for the $1|r_j| \sum w_j C_j$ problem are given in Hariri and Potts (1983). At time t , if $wT(S(t))$ is the total weighted tardiness of the partial schedule $S(t)$ and $LB'(U(t))$ is a lower bound value for the $1|r_j| \sum w_j C_j$ problem, then lb_2 gives a lower bound of the particular node such that $lb_2 = wT(S(t)) + LB'(U(t)) - \sum_{j \in U(t)} w_j d_j$.

4. The branch and bound algorithm

Since the problem is strongly NP-hard, enumerative algorithms such as the BB or Dynamic Programming (DP) approaches must be used to find an exact solution. For this problem, it may not be possible to develop efficient DP algorithms due to their time complexity. Hence we present a BB algorithm based on the dominance prop-

erties and two lower bounding procedures. A node at level k of the search tree corresponds to a partial sequence, P , in which jobs scheduled from the beginning of the schedule up to level k and a partial sequence, Q , of jobs scheduled at the end of the schedule. A node at level $k + 1$ of the tree is denoted as $P - i \cdots Q$, where P and Q are the defined partial sequences and job i is scheduled to $(k + 1)$ th position of the schedule. Since we have relatively weak but quickly computed lower bounds, our branching conditions rely heavily on the dominance properties proposed in Section 2.

On the other hand, the aim of a search strategy is to pick a candidate subproblem from the list of unfathomed candidate subproblems. One of the following two techniques can be used: (i) select the last unfathomed candidate subproblem added to the the active stack, AS , also known as the Depth First Search (DFS) method; or (ii) pick the unfathomed candidate subproblem whose corresponding lower bound is the smallest, also called the Best-First enumeration Scheme (BFS). To take advantage of both search methods, we use a hybrid approach of BFS and DFS as a search strategy. The maximum number of subproblems in the active stack is limited by a predetermined number. Until this size is reached, the BFS method is applied to get a general picture of solutions in all parts of the tree. Past this point, the proposed algorithm uses the DFS strategy because the complete schedule occurs at the bottom level of the search tree as described below.

The BB algorithm

- Step 0.* (Initialization) Set $seq \leftarrow 0$, $ub \leftarrow \infty$, $t^c \leftarrow r_{[1]}$, $S \leftarrow \emptyset$, $P \leftarrow \emptyset$, $Q \leftarrow \emptyset$, and $last \leftarrow n$. Calculate the breakpoint matrix. Determine $t_1 \leftarrow \max_{i \neq j \in J} \{t_{ij}^1, t_{ij}^2, t_{ij}^3\}$.
- Step 1.* (Global dominance) If any job $i \in A(t)$ satisfies Proposition 3 then set $S_{[seq]} \leftarrow i$, $t^c \leftarrow \max\{t^c + p_i, r_{\min}\}$, $P \leftarrow \{P - i\}$, and $seq \leftarrow seq + 1$. If $seq < last$ then repeat Step 1. Else if $wT_S < ub$, set $\bar{S} \leftarrow S$ and $ub \leftarrow wT_S$ then goto Step 7.
- Step 2.* (Global dominance) If any job $i \in U(t)$ satisfies Proposition 4 then set $S_{[last]} \leftarrow i$, $last \leftarrow last - 1$, and $Q \leftarrow \{i - Q\}$. If $seq < last$ then repeat Step 2. Else if $wT_S < ub$, set $\bar{S} \leftarrow S$ and $ub \leftarrow wT_S$ then goto Step 7.
- Step 3.* (Reduction to $1 || \sum w_j C_j$) If $t^c > t_1$ then schedule every unscheduled job $i \in U(t)$ in nonincreasing order of w_i/p_i . If $wT_S < ub$, set $\bar{S} \leftarrow S$ and $ub \leftarrow wT_S$ then goto Step 7.
- Step 4.* (Eliminating number of alternatives) If either $i \rightarrow j$ (from Propositions 8 or 1) or $i \prec j$ at time t^c for every unscheduled job $j \in U(t)$ (from Propositions 5, 6, 7, or 1) and $i \neq j$, then set $S_{[seq]} \leftarrow i$, $t^c \leftarrow \max\{t^c + p_i, r_{\min}\}$, $P \leftarrow \{P - i\}$, and $seq \leftarrow seq + 1$. If $seq < last$ then goto Step 1. Else if $wT_S < ub$, set $\bar{S} \leftarrow S$ and $ub \leftarrow wT_S$ then goto Step 7.

- Step 5.* (Selecting subproblem) For every unscheduled job $j \in U(t)$ which is not dominating $S_{[seq-1]}$ at t^c due to Propositions 5, 6, 7, 8, and 1 such that $r_j \leq \min\{t^c + p_{\min}, (r_k + p_k)_{k \in B(t^c)}\}$; where $p_{\min} = \min_{k \in A(t^c)}\{p_k\}$, let $lb_{P-j-Q} \leftarrow \max\{lb_1, lb_2\}$. If $lb_{P-j-Q} < ub$ for any unscheduled job $j \in U(t^c)$, insert it into active stack, AS, then store $t_{P-j-Q} \leftarrow \max\{t^c, r_j\}$ and $seq_{P-j-Q} \leftarrow seq$. Else goto Step 7.
- Step 6.* (Upper bounding) If $AS \neq \emptyset$ and $size - AS < Max - Stack - Size$ pick partial schedule $\{P - j \cdot Q\}$ with $\min lb_{P-j-Q}$ from AS (BFS). Else if $AS \neq \emptyset$ pick job j with LIFO rule (DFS). Set $S_{[seq]} \leftarrow j$, $t^c \leftarrow \max\{t^c + p_j, r_{\min}\}$, $P \leftarrow \{P - j\}$ and $seq \leftarrow seq + 1$. if $seq < last$ then goto step 1. Else if $wT_S < ub$, set $\bar{S} \leftarrow S$ and $ub \leftarrow wT_S$.
- Step 7.* (Branching) Eliminate all subsequences with $lb \geq ub$ from AS. If $AS \neq \emptyset$ and $size - AS < Max - Stack - Size$ pick partial schedule $\{P - j\}$ with $\min lb_{P-j-Q}$ from AS (BFS). Else if $AS \neq \emptyset$ pick job j with LIFO rule (DFS). Set $t^c \leftarrow t_{P-j-Q} + p_j$, $S_{[seq]} \leftarrow j$, $P \leftarrow \{P - j\}$ and $seq \leftarrow seq_{P-j-Q} + 1$ then goto Step 1.
- Step 8.* (Report optimum solution) Else report $S_{opt} \leftarrow \bar{S}$.

In the initialization step of the algorithm, the sequence is set to zero, the upper bound to infinity, the current time t^c , to the arrival of the first available job, and the current schedule S , P and Q are initialized to null. Alternatively, we can use the Apparent Tardiness Cost (ATC) dispatching rule to calculate an initial upper bound as discussed in Akturk and Yildirim (1998). The first step finds a job that globally dominates all the remaining unscheduled jobs at time t^c . If the schedule is not complete then we increment the current time t^c , either to the maximum of the completion time of job i or to the earliest release date of the unscheduled jobs r_{\min} . Before any new node is created, we check the proposed dominance rules to eliminate the dominated alternatives in Steps 2, 3 and 4. For each possible candidate partial sequence, both lb_1 and lb_2 are applied and the lower bound lb is set to their maximum value. If the lb is greater than or equal to the upper bound then this node is eliminated, else it is inserted to the active stack, AS. Their starting time and sequence are also recorded. In Step 6, if the tardiness of the current schedule wT_S is strictly less than the upper bound, S is kept as the incumbent schedule. In Step 7, a new subtree is grown by returning to Step 1 where the next partial schedule is selected according to the proposed hybrid search strategy. When the AS is empty the incumbent schedule \bar{S} is reported as optimal.

Consider the six job problem given in Table 1 as an example. The maximum stack size is determined to be five. The maximum breakpoint $t_l = 22$, hence the SWPT rule gives an optimum sequence for the remaining un-

Table 1. Job set parameters for example problem

Job	p_i	r_i	d_i	w_i
0	2	22	24	1
1	6	3	9	9
2	5	7	13	1
3	7	15	22	3
4	9	6	15	3
5	6	15	21	1

scheduled jobs for $t > 22$ as stated in Step 3. In order to demonstrate how the time dependent dominance rules are implemented in the BB algorithm, let us analyze the partial schedule at $t = 14$. Jobs 3, 4, and 5 are possible candidates, and $t_{2,4}^6 = 10/3$. So for $t \geq r_4 = 6$, job 4 precedes job 2, i.e., $4 \prec 2$. Therefore, scheduling job 4 right after job 2 will not lead to an optimal schedule due to Proposition 1. As shown in Fig. 3, the optimum schedule is $\{1 - 4 - 3 - 0 - 2 - 5\}$ and the minimum value of the total weighted tardiness is 57.

5. Computational analysis

To test the efficiency of the proposed BB algorithm, we implemented it using the Gnu C compiler with the -O2 optimizer option, run on a SPARC Station 10 under SunOS 5.4. The proposed algorithm was tested on a series of randomly generated problems as in Chu (1992b). For each job j , an integer processing time p_j and an integer weight w_j were generated from a discrete uniform distribution [1, 10]. Instead of finding due dates directly, we generated slack times between due dates and earliest completion times, i.e., $d_j - (p_j + r_j)$, from a uniform distribution between 0 and $\beta \sum_{j=1}^n p_j$ where three different β values [0.05, 0.25, 0.5] were used. Release dates r_j were generated from a uniform distribution ranging from 0 to $\alpha \sum_{j=1}^n p_j$, where four different α values [0.0, 0.5, 1.0, 1.5] were used. A total of 12 example sets were considered and 10 replications were taken for each combination, giving 120 randomly generated problems. The proposed algorithm was tested with 10, 15, and 20 jobs.

In the proposed hybrid search method, the maximum number of subproblems in the active stack is limited to 5000, which was determined after a number of test runs. There are two main performance measures for any BB algorithm; (i) the number of nodes considered to find an optimum solution; and (ii) the corresponding computation time. Whenever a problem was not solved within the upper limit of 4000 000 nodes, computation was abandoned for that problem. In Table 2, we present the results of the proposed algorithm for 10, 15 and 20 jobs along with the minimum, average, and maximum values of each measure. Table 2 also shows the influence of the α and β factors on problem difficulty. As anticipated, the prob-

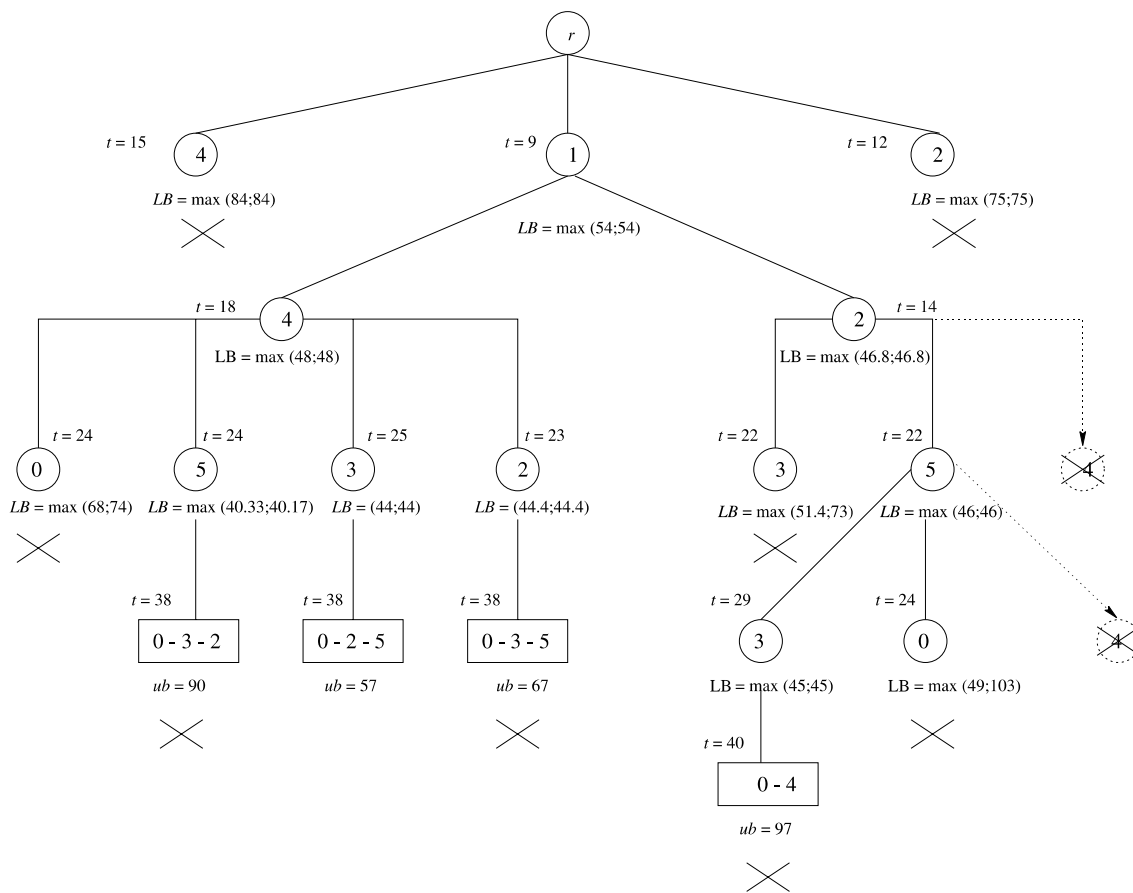


Fig. 3. BB tree for the example problem.

lems are most difficult when $\alpha = 0.5$ and β is between 0.25 and 0.5. The problem is relatively simple when α is small, because the release dates are not scattered. When $\alpha = 0.0$, the problem reduces to $1||\sum w_j T_j$, which is an easier class of problems, but still NP-hard. When α is large, the release dates are so scattered that there are a small number of active schedules. Since it is sufficient to consider active schedules, the search for an optimal solution among active schedules becomes easy. As a result, the algorithm is very fast for simple problem instances with loose due dates and loose release dates such as $\beta = 0.50$ and $\alpha \geq 1.0$, which usually resulted in non-tardy schedules. Similarly, for problem instances with tight release dates and due dates such as $\beta = 0.05$ and $\alpha = 0.0$, after scheduling first few jobs, all jobs become tardy and an optimal solution can be found in a short time due to Proposition 2.

For 15 job problems, there are $15! = 1.3 \times 10^{12}$ possible sequences. For the most difficult instance, namely $\alpha = 0.5$ and $\beta = 0.25$, the average number of nodes considered is 882 934.2, which is 1.5×10^6 times less than the number of sequences in a complete enumeration. The number of nodes required to find an optimum solution decreases to 14 for simpler cases, such as $\alpha = 0.0$ and $\beta = 0.05$. On the average, the number of nodes considered is 205 272. As expected, the computation time re-

quired for an optimal solution depends on the number nodes visited. For the harder problem instances, CPU times are on the average 1615.68 seconds, or 27 minutes. For the worst case of $\alpha = 0.5$ and $\beta = 0.5$, the CPU time is about 1 hour. For 20 job problems, limiting cases occurred when release dates or due dates are not so tight or loose. For the $(\alpha = 0.0, \beta = 0.5)$ and $(\alpha = 0.5, \beta = 0.5)$ cases, the algorithm exceeded the maximum node limit of 4000 000 and failed to find an optimum solution five and three times, out of 10 replications, respectively. Thus in these instances, the numbers given in Table 2 are lower bounds on the average number of nodes and the average computation time.

We used two different bounding procedures to calculate a lower bound at each node. These procedures are compared in Table 3 in terms of number of nodes that are eliminated for each example set along with the minimum, average and maximum values. The number of nodes considered is also provided in the table. Although none of the lower bounding algorithms dominates, the linear lower bound seems more efficient. Especially when the relative range of due dates is large, lb_2 provides a very weak bound since we subtract the weighted due dates. For 20 jobs, when $\alpha = 1.0$ and $\beta = 0.5$, lb_2 can eliminate at most 77 965 nodes, while lb_1 eliminates 92 902.5 nodes on the average

Table 2. Results of computational experiments

α	β	n	Number of nodes			CPU times		
			<i>min</i>	<i>avg</i>	<i>max</i>	<i>min</i>	<i>avg</i>	<i>max</i>
0.0	0.05	10	10	17.6	35	0	0.01	0.02
		15	14	52.4	123	0.05	0.15	0.43
		20	144	352.7	892	0.22	0.65	1.6
	0.25	10	110	352.5	1083	0.08	0.27	0.43
		15	5812	28 275	110 256	8.4	45.97	111.34
		20	508 709	1977 373.2	3997 319	181.78	1569.33	5377.5
	0.50	10	249	4027.8	8467	0.25	2.83	6.51
		15	44 605	552 206.9	1929 861	70.91	569.45	2151.05
		20	2317 148	2902 084	4000 000 ^a	2050.8	3071.33	4548.42
0.5	0.05	10	51	198	560	0.02	0.13	0.37
		15	1166	27 818.5	110 942	0.77	18.34	77.34
		20	58 074	712 961.4	1646 857	71.85	386.97	738.63
	0.25	10	98	1224.8	3433	0.06	1.07	3.28
		15	67 604	882 934.2	2332 666	29.09	365.86	869.46
		20	111 179	1129 594	3191 291	337.05	1765.66	4920.73
	0.50	10	122	1833.9	9103	0.1	2.09	10.36
		15	11 264	772 334.9	2022 782	50.60	1615.68	3728.18
		20	42	1107 718.1	4000 000 ^b	0.08	1348.15	4811.51
1.0	0.05	10	64	649.1	4405	0.02	0.26	1.7
		15	1075	3684.6	6976	0.95	4.97	12.18
		20	6029	115 840.5	766 128	9.11	160.92	822.83
	0.25	10	44	315.6	1318	0.03	0.23	1.15
		15	434	182 145.8	1603 228	0.63	535.92	3463.32
		20	48	593 646.4	3062 195	0.03	1107.02	3452.31
	0.50	10	9	292.3	2251	0	0.26	1.88
		15	17	946.8	5031	0.02	2.74	17.73
		20	18	443 117	1449 397	0.02	1336.74	3679.33
1.5	0.05	10	18	102.3	625	0.02	0.05	0.30
		15	81	410.6	939	0.04	0.23	0.77
		20	551	66 543.1	323 115	2.4	54.07	113.04
	0.25	10	12	68	134	0.02	0.05	0.10
		15	78	8620.9	66 380	0.06	30.26	293.01
		20	20	6781.5	50 538	0.03	37.83	248.90
	0.50	10	8	25.2	64	0	0.02	0.03
		15	16	3833.6	33 997	0	10.33	100.60
		20	19	61.1	148	0.02	0.05	0.08

a and b: 5 and 3 out of 10 replications could not be solved, respectively

and the number of nodes eliminated rises to 343 297 for the same case. The α value indicates the tightness of the relative range of release dates. Since the linear lower bound was originally developed for the $1 || \sum w_j T_j$ problem, we expect lb_1 to perform better when the α value is rather low. For 20 jobs, it eliminates 6722.8 nodes on the average for $\beta = 0.25$, when the average number of nodes considered is 1977 373.2. For this particular instance, lb_2 can eliminate at most 1188 nodes. When $\alpha = 1.0$ or $\alpha = 1.5$, lb_1 produces very weak bounds since it does not consider the release

dates of jobs. Therefore, when $\alpha = 1.0$ or $\alpha = 1.5$, lb_2 performs better than lb_1 . For $\alpha = 1.0$ and $\beta = 0.05$, lb_2 fathoms 5432 nodes on the average for the 20 job problem, while lb_1 can fathom only 695.1 nodes on the average, resulting in 115 840.5 nodes visited on the tree for searching an optimum solution.

When we compare the relative efficiencies of the proposed dominance properties, the most frequently used property is applying the SWPT rule to find an optimum sequence for the remaining jobs. This is especially true for

Table 3. Comparison of lower bounding procedures

α	β	n	Number of nodes			Number of nodes eliminated $LB1$			Number of nodes eliminated $LB2$		
			<i>min</i>	<i>avg</i>	<i>max</i>	<i>min</i>	<i>avg</i>	<i>max</i>	<i>min</i>	<i>avg</i>	<i>max</i>
0.0	0.05	10	10	17.6	35	0	0.3	3	0	0	0
		15	14	52.4	123	0	4.1	29	0	0.1	1
		20	144	352.7	892	0	7.8	17	0	0.2	2
	0.25	10	110	352.5	1083	9	29.6	80	0	0.2	2
		15	5812	28 275	110 256	22	392.3	1270	0	0.5	3
		20	508 709	1977 373.2	3997 319	0	6722.8	48 047	0	124.4	1188
	0.50	10	249	4027.8	8467	51	259	712	0	0.05	4
		15	44 605	552 206.9	1929 861	196	1168.8	5288	0	29.9	215
		20	2317 148	2902 084	4000 000*	0	3871.4	11 673	0	292.2	647
0.5	0.05	10	51	198	560	1	5.3	12	6	29.8	106
		15	1166	27 818.5	110 942	0	35.3	228	3	744.8	2269
		20	58 074	712 961.4	1646 857	0	159.5	655	416	5440.1	13 525
	0.25	10	98	1224.8	3433	3	30	95	0	24.4	66
		15	67 604	882 934.2	2332 666	0	4244.1	14 877	0	4985.6	20 778
		20	111 179	1129 594	3191 291	0	7954	65 950	0	1282	6999
	0.50	10	122	1833.9	9103	36	458.4	1271	2	48.3	144
		15	11 264	772 334.9	2022 782	0	9466.7	835 307	22	15 901.4	74 902
		20	42	1107 718.1	4000 000*	0	67 127.42	219 602	0	8543.71	34 415
1.0	0.05	10	64	649.1	4405	0	3.3	9	6	28.5	61
		15	1075	3684.6	6976	0	171.4	1461	215	715.9	1783
		20	6029	115 840.5	766 128	0	695.1	4437	13	5432	28 473
	0.25	10	44	315.6	1318	14	38.7	86	1	26	79
		15	434	182 145.8	1603 228	72	6460.2	25 332	53	2012.4	5736
		20	48	593 646.4	3062 195	0	78 999.5	292 332	0	15 535	58 342
	0.50	10	9	292.3	2251	0	38.1	207	0	45.9	280
		15	17	946.8	5031	0	773.7	4462	0	108.4	852
		20	18	443 117	1449 397	0	92 902.5	343 297	0	27 116	77 965
1.5	0.05	10	18	102.3	625	0	1	3	0	8	18
		15	81	410.6	939	0	31.7	132	1	66.2	173
		20	551	66 543.1	323 115	0	852.5	7938	93	1153.7	6954
	0.25	10	12	68	134	0	10.8	69	0	10.4	41
		15	78	8620.9	66 380	0	2323.1	20 909	0	2679.8	25 890
		20	20	6781.5	50 538	0	3017	15 370	0	151.4	774
	0.50	10	8	25.2	64	0	3	10	0	0.8	6
		15	16	3833.6	33 997	0	1919.3	18 993	0	1110.6	11 040
		20	19	61.1	148	0	2.2	12	0	0.7	4

* Maximum number of nodes considered in this study

$\beta = 0.05$ where due dates are tight, Proposition 2 seems to be the most efficient dominance property to eliminate a number of alternatives in the search tree. For bigger β values, i.e., $\beta = 0.25$ and $\beta = 0.5$, the impact of this property decreases gradually, although it is used again at lower levels of the tree. The average number of nodes fathomed using dominance properties are summarized in Table 4 for the 15 job case as an example. Proposition 7 is also quite effective in fathoming nodes in the algorithm. Especially for $\alpha = 0.5$, where most of the dominance rules

and lower bounding schemes perform weak, Proposition 7 eliminates jobs with small processing times but greater earliest completion times due to their release date. For $\alpha = 0.5$ and $\beta = 0.5$, the average number of nodes fathomed by Proposition 7 is 550 728.5, whereas other dominance properties eliminate around 150 000 nodes and lb_1 and lb_2 eliminate 9466.7 and 15 901.4 nodes, respectively on the average. Furthermore, the time dependent local dominance property stated in Proposition 1 is used frequently to fathom nodes in the BB tree.

Table 4. Comparison of the individual propositions when $n = 15$

α	β	Number of nodes eliminated						
		Proposition 1	Proposition 2	Proposition 3	Proposition 4	Proposition 5	Proposition 7	Proposition 8
0.0	0.05	2.3	47.7	0.6	0.2	0.6	2.9	0
	0.25	5351.8	24 232.6	326.9	121	4095	16 308.3	0
	0.50	145 782.8	287 373.1	56 444.8	224.7	82 113	731 882.2	3676
0.5	0.05	29 674.9	20 305.6	369.7	9.1	10 026.5	12 572.1	0.8
	0.25	359 225.3	605 379.1	41 510.4	33.5	79 044.4	431 632.8	0
	0.50	366 717.8	180 714.3	148 071.8	173	18 744.7	550 728.5	0
1.0	0.05	904.1	1501.1	383.3	65.8	608	829.5	0
	0.25	82 449.8	18 278.3	47 347	443.9	2728.3	172 280.2	0
	0.50	22	38.2	233	3	43	278.9	0
1.5	0.05	56.9	21.9	85.1	52.3	18	70.9	0
	0.25	1325.9	139.3	1751.6	0.8	412.3	2051.7	0
	0.50	106.3	0	808.5	13.4	783.7	838.1	0

6. Concluding remarks

We propose a set of new dominance properties for the $1|r_j|\sum w_jT_j$ problem that can be utilized in any exact approach. We also enlarged the region for which the $1|r_j|\sum w_jT_j$ problem can be solved optimally by the SWPT rule. We tested the proposed BB algorithm on a set of randomly generated problems. Even for 10 jobs, commercial optimization software packages, like CPLEX, fail to find an optimum solution. Therefore, any method that can solve even a 15 job problem will be a contribution to the literature. We also know that the $1|r_j|\sum T_j$ problem can be solved with up to 30 jobs (Chu, 1992b), while for the equal release dates problem, $1||\sum T_j$, can be solved for more than 300 jobs. Furthermore, BB algorithms for the $1||\sum w_jT_j$ and $1|r_j|\sum w_jC_j$ problems can solve problem instances with up to 40 jobs (Potts and van Wassenhove, 1985; Beloudah *et al.*, 1992). To the best of our knowledge, this study is the first exact approach that simultaneously deals with the weighted tardiness and unequal release dates problems on a single machine. For further research, better lower bounding procedures would improve the proposed BB algorithm for larger size problems.

Acknowledgments

The authors would like to thank the Department Editor Prof. Reha Uzsoy and two anonymous referees whose constructive comments have been used to improve this paper.

References

Abdul-Razaq, T.S., Potts, C.N. and van Wassenhove, L.N. (1990) A survey of algorithms for the single-machine total weighted

tardiness scheduling problem. *Discrete Applied Mathematics*, **26**, 235–253.

Akturk, M.S. and Ozdemir, D. (1998) New dominance properties for $1|r_j|\sum w_jT_j$ problem. Technical Report No. 98-30, Department of Industrial Engineering, Bilkent University, Turkey.

Akturk, M.S. and Yildirim, M.B. (1998) A new lower bounding scheme for the total weighted tardiness problem. *Computers and Operations Research*, **25**, 265–278.

Beloudah, H., Posner, M.E. and Potts, C.N. (1992) Scheduling with release dates on a single machine to minimize total weighted completion time. *Discrete Applied Mathematics*, **36**, 213–231.

Chand, S., Traub, R. and Uzsoy, R. (1996) Single machine scheduling with dynamic arrivals: decomposition results and an improved algorithm. *Naval Research Logistics*, **43**, 709–719.

Chu, C. (1992a) A branch-and-bound algorithm to minimize total flow time with unequal release dates. *Naval Research Logistics*, **39**, 859–875.

Chu, C. (1992b) A branch-and-bound algorithm to minimize total tardiness with unequal release dates. *Naval Research Logistics*, **39**, 265–283.

Chu, C. and Portmann, M.C. (1992) Some new efficient methods to solve the $n|1|r_j|\sum T_i$ scheduling problem. *European Journal of Operational Research*, **58**, 404–413.

Emmons, H. (1969) One machine sequencing to minimize certain functions of job tardiness. *Operations Research*, **17**, 701–715.

Hariri, A.M.A. and Potts, C.N. (1983) An algorithm for single machine sequencing with release dates to minimize total weighted completion time. *Discrete Applied Mathematics*, **5**, 99–109.

Lawler, E.L. (1977) A ‘pseudopolynomial’ algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, **1**, 331–342.

Potts, C.N. and van Wassenhove, L.N. (1985) A branch and bound algorithm for total weighted tardiness problem. *Operations Research*, **33**, 363–377.

Potts, C.N. and van Wassenhove, L.N. (1988) Algorithms for scheduling a single machine to minimize the weighted number of late jobs. *Management Science*, **34**, 843–858.

Rachamadugu, R.M.V. (1987) A note on weighted tardiness problem. *Operations Research*, **35**, 450–452.

Rinnooy Kan, A.H.G. (1976) *Machine Scheduling Problems: Classification, Complexity and Computations*, Nijhoff, The Hague.

Rinnooy Kan, A.H.G., Lageweg, B.J. and Lenstra, J.K. (1975) Minimizing total costs in one-machine scheduling. *Operations Research*, **23**, 908–927.

- Smith, W.E. (1956) Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, **3**, 59–66.
- Szwarc, W. (1993) Adjacent orderings in single machine scheduling with earliness and tardiness penalties. *Naval Research Logistics*, **40**, 229–243.
- Vairaktarakis, G.L. and Lee, C.Y. (1995) The single-machine scheduling problem to minimize total tardiness subject to minimum number of tardy jobs. *IIE Transactions*, **27**, 250–256.

Biographies

M. Selim Akturk is an Assistant Professor of Industrial Engineering at Bilkent University, Turkey. He holds a Ph.D. in Industrial Engineering

from Lehigh University, USA, and B.S.I.E. and M.S.I.E. from Middle East Technical University, Turkey. His current research interests include hierarchical planning of large-scale systems, production scheduling, cellular manufacturing systems, and advanced manufacturing technologies. Dr. Akturk is a senior member of IIE and member of INFORMS.

Deniz Ozdemir is a Ph.D. student in the Production and Operations Management Programme at INSEAD, France. She received B.S.I.E. and M.S.I.E. degrees from Bilkent University, Turkey. Her current research interests include production scheduling and applied optimization.

Contributed by the Scheduling Department.