A Branch-and-Bound Algorithm for the Single Machine Earliness and Tardiness Scheduling Problem

Ching-Fang Liaw

Department of Industrial Engineering and Management
Chaoyang University of Technology
Taichung, Taiwan

E-mail: cfliaw@mail.cyut.edu.tw
Fax: 886-4-3742327

August 17, 1998

* This research was supported by the National Science Council (NSC) of Republic of China under grant NSC 88-2213-E-324-011.
1. INTRODUCTION

The single machine earliness and tardiness scheduling 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 on a single machine that can handle at most one job at a time. The machine and the jobs are assumed to be continuously available from time zero onwards. Job $J_i$, $i = 1, 2, \ldots, n$, requires an integral uninterrupted processing time $p_i$ and should ideally be completed exactly on its due date $d_i$. Given a schedule, the earliness of $J_i$ is defined as $E_i = \max\{0, d_i - C_i\}$ and the tardiness of $J_i$ is defined as $T_i = \max\{0, C_i - d_i\}$, where $C_i$ is the completion time of $J_i$, $i = 1, 2, \ldots, n$. The objective is to find a schedule that minimizes the sum of weighted earliness and weighted tardiness $\sum_{i=1}^{n}(\alpha_i E_i + \beta_i T_i)$ subject to the constraint that no machine idle time is allowed, where $\alpha_i$ and $\beta_i$ are the earliness and tardiness weights for $J_i$.

The inclusion of both the early and tardy costs in the objective function captures the just-in-time (JIT) production philosophy, which emphasizes producing goods only when they are necessary. In general, the early cost can be regarded as a holding cost for finished goods, deterioration of perishable goods and opportunity costs. The tardy cost can be considered as the backlogging cost, which includes performance penalties, lost sales and lost goodwill. The assumption of no machine idle time
represents a type of production environment where the machine cost of being kept idle is larger than the early cost incurred by completing a job before its due date, or the machine is heavily loaded, so that it is kept running and no idle time is permitted.

The problem is known to be NP-hard in the strong sense, since the special case where $\alpha_i = 0, i = 1, 2, \ldots, n$, is known to be such [1, 2]. A large number of studies have been done on the single machine scheduling problem involving both job earliness and tardiness. We only review in the following the papers examining the problem which is exactly the same or closely related to ours, that is, machine idle time is prohibited, jobs have distinct due date and distinct earliness and tardiness weights, and the objective is to minimize the sum of weighted earliness and weighted tardiness. The readers are referred to Baker and Scudder[3] for an overview, and some recent studies (see, for example, [4, 5, 6, 7]) for more information on the single machine earliness and tardiness scheduling problem.

Abdul-Razaq and Potts[8] developed a branch-and-bound algorithm with lower bounds obtained by the dynamic programming state-space relaxation technique which maps the original state-space onto a smaller state-space and performs the recursion on this smaller state-space. The lower bound is further improved through the use of job penalties and the use of state-space modifiers. Their computational results suggest that problems containing more than 25 jobs may lead to excessive solution times.
Ow and Morton\cite{9} presented priority dispatching rules and a filtered beam search method which uses a priority function for evaluation purposes to obtain near optimal solutions. Their computational studies show that the filtered beam search method can produce very good solutions with a relatively small search tree. Li\cite{10} proposed a neighborhood search heuristic procedure as well as a branch-and-bound algorithm. The branch-and-bound algorithm is based on a decomposition of the problem into two subproblems and two efficient multiplier adjustment methods for solving two Lagrangian dual subproblems. Their computational tests indicate that the neighborhood search heuristic procedure is not only efficient but also robust in producing near optimal solutions for large problems, and the branch-and-bound algorithm can efficiently obtain optimal solutions for small problems.

Szwarc\cite{11} examined a special case of our problem where $\alpha_i = \alpha$ and $\beta_i = \beta$ for all $i=1, 2, \ldots, n$. Based on both conditional and unconditional orderings of two adjacent jobs, they developed a branching scheme that can be incorporated in any branch-and-bound algorithm once a lower bound is found. Without the lower bound the branching scheme can solve only small problems. Azizoglu, Kondakci and Kirca\cite{12} assumed that $\alpha_i = \alpha$ and $\beta_i = 1 - \alpha$ for all $i=1, 2, \ldots, n$, and gave a branch-and-bound algorithm together with efficient lower and upper bounds. Their computational studies indicate that the branch-and-bound algorithm performs well for
problems with up to 20 jobs. Yano and Kim[13] presented a branch-and-bound algorithm and a heuristic procedure for the problem where the earliness and tardiness weights are proportional to the processing times of the jobs, i.e., $\alpha_i = ap_i$ and $\beta_i = bp_i$ for all $i=1, 2, \ldots, n$ where $a$ and $b$ are nonnegative real numbers. Their computational tests show that their lower bounds are not very strong and hence the proposed branch-and-bound algorithm can solve only small problems.

In this paper, we develop efficient lower and upper bounding procedures. The lower bounding procedure is based on a Lagrangian relaxation that decomposes the problem into two subproblems: a total weighted completion time subproblem and a slack variable subproblem. A lower bound of the total weighted completion time subproblem is obtained by a multiplier adjustment method that can be computed in $O(n)$ time. A lower bound of the slack variable subproblem is computed using a single pass procedure also in $O(n)$ time. The sum of these two lower bounds of the subproblems gives rise to the final lower bound for the problem. The upper bounding procedure is based on a two-phase heuristic procedure. In the first phase, an initial schedule is constructed using a priority dispatching rule from Ow and Morton[9]. In the second phase, a local improvement procedure is used to improve the initial schedule. The lower and upper bounding procedures, together with some dominance rules, are used in a branch-and-bound scheme. The branch-and-bound
algorithm is tested on problems with up to 50 jobs. Computational results show that our lower and upper bounds are very tight, and the algorithm performs very well.

This paper is organized as follows. Section 2 describes the derivation of the lower bound. Two simple dominance rules are presented in section 3. An efficient heuristic procedure is developed in section 4. The implementation of the branch-and-bound algorithm is discussed in section 5. Computational experiments are provided in section 6, which are followed by some concluding remarks in section 7.

2. DERIVATION OF THE LOWER BOUND

In this section, we formulate the problem, decompose the problem into two subproblems based on Lagrangian relaxation, and develop an efficient multiplier adjustment method to compute the values of the Lagrangian multipliers. The single machine earliness and tardiness scheduling problem can be logically formulated as problem (P):

\[ Z = \min \sum_{i=1}^{n} (\alpha_i E_i + \beta_i T_i) \]

s.t. \[ T_i = C_i - d_i + E_i \quad i = 1, 2, \ldots, n \quad (1) \]

\[ T_i \geq 0 \quad i = 1, 2, \ldots, n \quad (2) \]

\[ E_i \geq 0 \quad i = 1, 2, \ldots, n \quad (3) \]

the capacity and availability constraints of the machine \quad (4)

where constraints (1), (2) and (3) reflect the definitions of job earliness and tardiness.
A Lagrangian relaxation of constraints (1) yields the Lagrangian problem (LR):

\[
\begin{align*}
L(\lambda) &= \min \sum_{i=1}^{n} [\lambda_i (C_i - d_i) + (\alpha_i + \lambda_i) E_i + (\beta_i - \lambda_i) T_i] \\
s.t. & \quad (2), (3) \text{ and } (4)
\end{align*}
\]

where \( \lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n) \) is the vector of corresponding Lagrangian multipliers.

From Lagrangian relaxation theory, for any choice of \( \lambda \), \( L(\lambda) \) is a lower bound for \( Z \). Since constraints (2) and (3) affect only the last two components of the Lagrangian objective function and constraints (4) only the first one, (LR) is decomposed into the following two subproblems:

\[
\begin{align*}
L_1(\lambda) &= \min \sum_{i=1}^{n} \lambda_i (C_i - d_i) \\
s.t. & \quad \text{the capacity and availability constraints of the machine}
\end{align*}
\]

and

\[
\begin{align*}
L_2(\lambda) &= \min \sum_{i=1}^{n} [(\alpha_i + \lambda_i) E_i + (\beta_i - \lambda_i) T_i] \\
s.t. & \quad E_i \geq 0 \quad i = 1, 2, \ldots, n \\
& \quad T_i \geq 0 \quad i = 1, 2, \ldots, n
\end{align*}
\]

If \( \alpha_i + \lambda_i < 0 \) or \( \beta_i - \lambda_i < 0 \) for some \( i, i = 1, 2, \ldots, n \), then we have \( E_i = \infty \) or \( T_i = \infty \), giving \( L_2(\lambda) = -\infty \). To avoid this, we therefore require that \(-\alpha_i \leq \lambda_i \leq \beta_i\) for each \( i, i = 1, 2, \ldots, n \). We are interested in finding or approximating the vector \( \lambda^* \) of Lagrangian multipliers that solves the Lagrangian dual problem (D):

\[
\begin{align*}
\max & \quad L(\lambda) \\
s.t. & \quad -\alpha_i \leq \lambda_i \leq \beta_i \quad i = 1, 2, \ldots, n
\end{align*}
\]

The Lagrangian subproblem (LR1) is solved by scheduling the jobs in nonincreasing order of \( \lambda_i / p_i \) according to Smith’s rule[14]. The Lagrangian
multipliers can be iteratively updated using the subgradient optimization method to
improve the lower bound[15]. Since the subgradient optimization method is slow in
practice, we resort to the multiplier adjustment method. In general, the multiplier
adjustment method is a problem specific approximation method that exploits the
structure of the problem and the formulation. A multiplier adjustment method is
generally much faster than the subgradient optimization method, but it cannot be
guaranteed to obtain lower bounds that are as good. However, many successful
applications of multiplier adjustment method [17, 18, 19, 20, 21, 22] have indicated
that the gain in speed over the subgradient optimization method compensates the
possible loss in lower quality more than sufficiently.

The Lagrangian subproblem \((LR_2)\) is solved by setting \(E_i = T_i = 0\) for each \(i, i = 1, 2, \ldots, n\), resulting in \(L_2(\lambda) = 0\). Although it seems that the subproblem \((LR_2)\) does
not contribute to the lower bound \(L(\lambda)\), we retain it; as we will see later, we gain by
it since in a nontrivial problem we can not have \(E_i = T_i = 0\) for each \(i, i = 1, 2, \ldots, n\).

2.1 A multiplier adjustment method

In this section, we extend the multiplier adjustment method developed by Potts
and van Wassenhove[22] to compute values of the Lagrangian multipliers for
subproblem \((LR_1)\). The multipliers are then used in subproblem \((LR_2)\) to improve the
lower bound. The multiplier adjustment method first requires a heuristic to
sequence the jobs and then chooses the multipliers so that the resulting lower bound is as large as possible. Assume that the jobs are renumbered so that the sequence generated by the heuristic selected is \((1, 2, \ldots, n)\). In addition, let \(C^*_i\) be the completion time of job \(i, i = 1, 2, \ldots, n\), in the sequence. Finally, the multiplier adjustment method requires the sequence generated \((1, 2, \ldots, n)\) along with the corresponding completion times \(C^*_i, i = 1, 2, \ldots, n\), to be a solution of the subproblem \((LR_1)\). Therefore, the multipliers are chosen subject to the constraints

\[
\lambda_i / p_i \geq \frac{\lambda_{i+1}}{p_{i+1}} \quad i = 1, 2, \ldots, n-1
\]

To obtain the maximum value of \(L_1(\lambda)\), we solve the problem \((LD(C^*))\):

\[
W = \max \sum_{i=1}^{n} \lambda_i (C^*_i - d_i)
\]

\[
(LD(C^*)) \quad \text{s.t.} \quad \frac{\lambda_i}{p_i} \geq \frac{\lambda_{i+1}}{p_{i+1}} \quad i = 1, 2, \ldots, n-1 \quad (5)
\]

\[-\alpha_i \leq \lambda_i \leq \beta_i \quad i = 1, 2, \ldots, n \quad (6)
\]

**Lemma 1:** Constraints (6) can be replaced with

\[-\alpha^*_i \leq \lambda_i \leq \beta^*_i \quad i = 1, 2, \ldots, n \quad (7)
\]

without changing the solution of problem \((LD(C^*))\), where

\[-\alpha^*_i = p_i \max \{-\alpha_k / p_k : k = i, i+1, \ldots, n}\] and \(\beta^*_i = p_i \min \{\beta_k / p_k : k = 1, 2, \ldots, i\}\).

Proof: Suppose that for any job \(i\), we have \(\beta^*_i = p_i \beta_h / p_h\) for some \(h, 1 \leq h \leq i\).

Then, by definition, \(\beta^*_h = \beta_h\). From (5) and (6), \(\lambda_i / p_i \leq \lambda_h / p_h \leq \beta_h / p_h\), which
implies that $\lambda_i \leq p_i \beta_h / p_h = \beta_i^*$. Similarly, suppose that for any job $i$, we have

$$-\alpha_i^* = p_i (-\alpha_h / p_h)$$

for some $h$, where $i \leq h \leq n$. By definition, we have $\alpha_h^* = \alpha_h$. From (5) and (6), $\lambda_i / p_i \geq \lambda_h / p_h \leq -\alpha_h / p_h$, which implies that $\lambda_i \geq -\alpha_i^*$. By definition, $\beta_i^* \leq \beta_i$ and $\alpha_i^* \leq \alpha_i$ for $i = 1, 2, \ldots, n$, constraints (6) therefore can be replaced with (7).

Note that if the jobs are sequenced by the WSPT (weighted shortest processing time) rule using $\beta_i$ as the weight of job $i$, for each $i = 1, 2, \ldots, n$, i.e.,

$$\beta_1 / p_1 \geq \beta_2 / p_2 \geq \ldots \geq \beta_n / p_n,$$

then we have $\beta_i^* = \beta_i$ for all $i = 1, 2, \ldots, n$. On the other hand, if the jobs are sequenced by the WLPT (weighted longest processing time) rule using $\alpha_i$ as the weight of job $i$, $i = 1, 2, \ldots, n$, i.e.,

$$\alpha_1 / p_1 \leq \alpha_2 / p_2 \leq \ldots \leq \alpha_n / p_n,$$

then we have $\alpha_i^* = \alpha_i$ for all $i = 1, 2, \ldots, n$.

We now present a procedure which is an extension of the multiplier adjustment method in [22] for solving problem $(LD(C^*))$. Compute $L_i^* = C_i^* - d_j$ and

$$V_i = \sum_{k=1}^{j} p_k L_k^*$$

for $i = 1, 2, \ldots, n$. Set $V_0 = 0$, $s_0 = 0$ and $S = \{ s_0 \}$. Compute the set $S = \{ s_0, s_1, \ldots, s_r \}$ of values as follows. Having found $s_0, s_1, \ldots, s_{k-1}$, choose $s_k$ as small as possible so that $s_k > s_{k-1}$ and so that $V_{s_k} > V_{s_{k-1}}$. Let $s_r$ be the largest number in $S$. Note that $S = \{ s_0, s_1, \ldots, s_r \}$ is an ordered integer set with its elements in ascending order of their values. In essence, $S$ is the set of those jobs that each has a positive contribution to the maximum value $W$. Therefore, the larger $\lambda_i$
is for \( i \in S \), the larger is \( W \). Compute 
\[
U_i = \sum_{k=i}^n p_k L_k^* \quad \text{for } i = s_r + 1, s_r + 2, \ldots, n.
\]

Set \( U_{n+1} = 0 \), \( s_{n+1}^* = n + 1 \) and \( S^* = \{ s_{n+1}^* \} \). The set \( S^* = \{ s_{n+1}^*, \ldots, s_{l+1}^*, s_l^* \} \) of
values is computed as follows. Having found \( s_{n+1}^*, \ldots, s_{k+2}^*, s_{k+1}^* \), choose \( s_k^* \) as
large as possible so that \( s_k^* < s_{k+1}^* \) and so that \( U_{s_k} < U_{s_{k+1}} \). Similarly,
\[
S^* = \{ s_{n+1}^*, \ldots, s_{l+1}^*, s_l^* \}
\]
is an ordered integer set with its elements in descending order
of their values. However, each job \( i \in S^* \) has a negative contribution to the
maximum value \( W \). Therefore, the smaller \( \lambda_i \) is for \( i \in S^* \), the larger is \( W \).

Finally, the sets \( S \) and \( S^* \) are used to determine the values of the multipliers, as shown
in the next result.

**Theorem 1:** Problem \((LD(C^*))\) is solved by setting \( \lambda_i = \lambda_i^* \) for \( i = 1, 2, \ldots, n \), where
\[
\lambda_i^* = -\alpha_i^* \quad \text{if } i \in S^* \tag{8}
\]
\[
\lambda_i^* = \lambda_{i-1}^*(p_i / p_{i-1}) \quad \text{if } i > s_r \text{ and } i \notin S^* \tag{9}
\]
\[
\lambda_i^* = \beta_i^* \quad \text{if } i \in S \tag{10}
\]
\[
\lambda_i^* = \lambda_{i+1}^*(p_i / p_{i+1}) \quad \text{if } i \leq s_r \text{ and } i \notin S \tag{11}
\]

Proof: Consider any optimal solution \( \lambda_i = \lambda_i^* \) for \( i = 1, 2, \ldots, n \) to problem \((LD(C^*))\)
in which \( \lambda_i^* \) for \( i = 1, 2, \ldots, n \) are chosen so that \( \sum_{i=1}^n \lambda_i^* \) is as large as possible.

We first prove by contradiction that \( \lambda_i^* = \lambda_{i-1}^*(p_i / p_{i-1}) \) for \( i = s_r^* + 1, s_r^* + 2, \ldots, n \)
and \( i \notin S^* \). Suppose that \( k \) is the largest number in \{\( s_r^* + 1, s_r^* + 2, \ldots, n \)\} such that
\( \lambda_k^* \neq \lambda_{k-1}^*(p_k / p_{k-1}) \). Let \( \delta = \lambda_k^* / p_k - \lambda_{k-1}^* / p_{k-1} \). Now since
\[ \delta + (\lambda_{k-1}^* / p_{k-1}) = \lambda_k^* / p_k = \ldots = \lambda_n^* / p_n , \] 
\[ \sum_{i=k}^n p_i L_i^* \geq 0 \quad \text{and} \quad \delta < 0 , \]
we have
\[ \sum_{i=1}^n \lambda_i^* L_i^* = \sum_{i=1}^{k-1} \lambda_i^* L_i^* + \sum_{i=k}^n [\delta + (\lambda_{k-1}^* / p_{k-1})] p_i L_i^* \]
\[ \leq \sum_{i=1}^{k-1} \lambda_i^* L_i^* + \sum_{i=k}^n (\lambda_{k-1}^* / p_{k-1}) p_i L_i^* \]

Setting
\[ \lambda_i^* = \begin{cases} 
\lambda_i^* & \text{if } i = 1, 2, \ldots, k-1 \\
 p_i (\lambda_{k-1}^* / p_{k-1}) & \text{if } i = k, k+1, \ldots, n 
\end{cases} \]
we have \[ \sum_{i=1}^n \lambda_i^* L_i^* \leq \sum_{i=1}^n \lambda_i^* L_i^* . \] Thus, \( \{ \lambda_1^*, \lambda_2^*, \ldots, \lambda_n^* \} \) is also an optimal solution to problem \( (LD(C^*)) \). Moreover, \[ \sum_{i=1}^n \lambda_i^* L_i^* < \sum_{i=1}^n \lambda_i L_i^* \], contradicting the assumption that \[ \sum_{i=1}^n \lambda_i L_i^* \] is maximal. Therefore, \[ \lambda_i^* = \lambda_{i-1}^* (p_i / p_{i-1}) \] for \( i = s_1^* + 1, s_1^* + 2, \ldots, n \) and consequently, \[ \lambda_i^* = p_i (\lambda_{s_1^*}^* / p_{s_1^*}) \] for \( i = s_1^* + 1, s_1^* + 2, \ldots, n \).

We next show by contradiction that \[ \lambda_{s_1^*}^* = -\alpha_{s_1^*}^* . \] Suppose \[ \lambda_{s_1^*}^* > -\alpha_{s_1^*}^* . \] Then,
\[ \sum_{i=1}^n \lambda_i^* L_i^* = \sum_{i=1}^{s_1^* - 1} \lambda_i^* L_i^* + \sum_{i=s_1^*}^n p_i L_i^* (\lambda_{s_1^*}^* / p_{s_1^*}) > \sum_{i=1}^{s_1^* - 1} \lambda_i^* L_i^* + \sum_{i=s_1^*}^n p_i L_i^* (-\alpha_{s_1^*}^* / p_{s_1^*}) . \]
The solution \( \lambda_i = \lambda_i^* \) for \( i = 1, 2, \ldots, s_1^* - 1, \) and \( \lambda_i = p_i (-\alpha_{s_1^*}^* / p_{s_1^*}) \) for \( i = \) \( s_1^* + 1, \ldots, n \), is a feasible solution since
\[ \lambda_{s_1^* - 1}^* / p_{s_1^* - 1} \geq -\alpha_{s_1^* - 1}^* / p_{s_1^* - 1} \geq -\alpha_{s_1^*}^* / p_{s_1^*} \] . Moreover, this solution has a larger objective value. Hence, \[ \lambda_{s_1^*}^* = -\alpha_{s_1^*}^* . \]

Since the choice of \( \lambda_{s_1^*}, \lambda_{s_1^* + 1}, \ldots, \lambda_n \) does not restrict the choice of \( \lambda_i \) for \( i = 1, 2, \ldots, s_1^* - 1, \) the argument is repeated, thus establishing (8) and (9). For the argument to establish (10) and (11), see the proof of Theorem 1 in [22]. \( \Box \)

**Theorem 2**: If the jobs are sequenced by the WSPT rule using \( \beta_i \) as the weight of
job \( i, i = 1, 2, \ldots, n \), and the resulting schedule contains no early jobs, i.e., \( C_i^* - d_i \geq 0 \) for all \( i = 1, 2, \ldots, n \), then \( W = Z \).

Proof: It is known that the WSPT rule minimizes total weighted lateness. If such a schedule does not have any early jobs, then the earliness cost of the schedule is zero and the total weighted lateness of the schedule is also the minimum total weighted earliness and tardiness cost. That is, \( \sum_{i=1}^{n} \beta_i (C_i^* - d_i) = Z \). In this case, from Theorem 1, we have \( \lambda_i = \beta_i \) for all \( i = 1, 2, \ldots, n \) such that \( C_i^* - d_i > 0 \).

Therefore, we have

\[
W = \sum_{i=1}^{n} \lambda_i (C_i^* - d_i) = \sum_{i=1}^{n} \beta_i (C_i^* - d_i) = Z .
\]

**Theorem 3:** If the jobs are sequenced by the WLPT rule using \( \alpha_i \) as the weight of job \( i, i = 1, 2, \ldots, n \), and the resulting schedule contains no tardy jobs, i.e., \( C_i^* - d_i \leq 0 \) for all \( i = 1, 2, \ldots, n \), then \( W = Z \).

Proof: It is known that the WLPT rule maximizes total weighted lateness. In doing so, total weighted earliness is also minimized. If such a schedule does not have any tardy jobs, then the tardiness cost of the schedule is zero and the total weighted earliness of the schedule is also the minimum total weighted earliness and tardiness cost. That is, \( \sum_{i=1}^{n} \alpha_i (d_i - C_i^*) = Z \). In this case, by Theorem 1, we have

\[
\lambda_i^* = -\alpha_i^* = -\alpha_i \text{ for all } i = 1, 2, \ldots, n \text{ such that } C_i^* - d_i < 0 .
\]

Therefore, we have

\[
W = \sum_{i=1}^{n} \lambda_i^* (C_i^* - d_i) = \sum_{i=1}^{n} (-\alpha_i)(C_i^* - d_i) = \sum_{i=1}^{n} \alpha_i (d_i - C_i^*) = Z .
\]

Theorems 2 and 3 give the conditions under which the lower bound value
produced by the multipliers adjustment method equals the optimum objective value of $(P)$. Theorem 2 also implies that when most jobs of a problem are tardy, the multiplier adjustment method with jobs sequenced by the WSPT rule using $\beta_i$ as the weight of job $i$ for all $i = 1, 2, \ldots, n$ can be used to produce a tight lower bound for $(P)$. Similarly, Theorem 3 also implies that when most jobs of a problem are early, the multiplier adjustment method with jobs sequenced by the WLPT rule using $\alpha_i$ as the weight of job $i$ for all $i = 1, 2, \ldots, n$ can be used to produce a tight lower bound for $(P)$. The multiplier adjustment method requires $O(n)$ time if jobs are preordered according to the chosen heuristic, and hence is very efficient. In some cases, it generates a very tight lower bound for problem $(P)$.

2.2 Improving the lower bound

In this section, we present a procedure for improving the Lagrangian lower bound by solving subproblem $(LR_2)$ using the multipliers $(\lambda^*)$ obtained from Theorem 1. Note that subproblem $(LR_2)$ has the same structure as the modified slack variable problem given by Hoogeveen and van de Velde[23], and hence the solution procedure proposed in [23] can be used to solve this problem. The subproblem $(LR_2)$ has the same computational complexity as problem $(P)$, and therefore we try to compute a lower bound for $L_2(\lambda^*)$, and then add it to the Lagrangian lower bound $L_1(\lambda^*)$.

If it is feasible to process each job $J_i$ in the interval $[d_i - p_i, d_i]$, no tardiness
or earliness costs will be incurred. However, in this case, the original problem \( (P) \) would be trivial. Suppose that there are two jobs \( J_i \) and \( J_k \) for which the ideal processing intervals \( [d_i - p_i, d_i] \) and \( [d_k - p_k, d_k] \) overlap. Jobs \( J_i \) and \( J_k \) are called conflicting jobs, since \( J_i \) or \( J_k \) will be early or tardy. The minimum penalty to settle the conflict is obtained by evaluating the following four cases:

1. scheduling \( J_i \) and \( J_k \) in the interval \( [d_i - p_i, d_i + p_k] \) with \( J_i \) before \( J_k \),
2. scheduling \( J_i \) and \( J_k \) in the interval \( [d_k - p_k - p_i, d_k] \) with \( J_i \) before \( J_k \),
3. scheduling \( J_i \) and \( J_k \) in the interval \( [d_k - p_k, d_k + p_i] \) with \( J_k \) before \( J_i \),
4. scheduling \( J_i \) and \( J_k \) in the interval \( [d_i - p_i - p_k, d_i] \) with \( J_k \) before \( J_i \).

All other cases are dominated by these four cases. The minimum penalty is readily computed since the minimum penalty of scheduling \( J_i \) before \( J_k \) is equal to

\[
\min \{\alpha_i + \lambda_i^* + \beta_k - \lambda_k^* (d_i + p_k - d_k)\},
\]

and the minimum penalty of scheduling \( J_k \) before \( J_i \) is equal to

\[
\min \{\alpha_k + \lambda_k^* + \beta_i - \lambda_i^* (d_k + p_i - d_i)\}.
\]

A lower bound for subproblem \((LR_2)\) can be computed as follows. First, we sequence the jobs in nondecreasing order of the due dates, and renumber them.
accordingly. Then, we identify pairs of adjacent conflicting jobs; no job can appear in more than one pair. Finally, we compute for each pair the minimum penalty required to settle the conflict. The sum of these penalties is a lower bound of \( L_2(\lambda) \). Adding this sum to \( L_1(\lambda) \) gives rise to the final lower bound \( L(\lambda) \).

Note that this lower bound of \( L_2(\lambda) \) is computed in \( O(n) \) time if the jobs are prearranged.

In our implementation, instead of computing a lower bound based on specifying pairs of adjacent conflicting jobs, we compute in a similar way a lower bound of \( L_2(\lambda) \) by specifying triples of adjacent conflicting jobs. This strategy has produced in average slightly stronger lower bounds in initial tests. To compute the minimum penalty for settling the conflict in such a triple, we evaluate eighteen cases. The sum of these penalties is added to \( L_1(\lambda) \) to obtain our final lower bound \( L(\lambda) \).

### 3. DOMINANCE RULES

This section gives two simple dominance rules for the problem under consideration. Dominance rules are basically necessary conditions for any optimal schedule that can be generated. Applying such rules results in a set of precedence relations between jobs. These precedence relations are then used to reduce the number of branches in a branch-and-bound search tree. In the following, Theorem 4
is a result from Ow and Morton[9], and Theorem 5 is simple dominance rule
developed here.

Theorem 4: (The adjacency condition) All adjacent pairs of jobs in an optimal
schedule must satisfy the following condition

$$\beta_i p_k - \Omega_{ik}(\beta_i + \alpha_i) \geq \beta_k p_i - \Omega_{ki}(\beta_k + \alpha_k)$$

where job $i$ immediately precedes job $k$, and $\Omega_{ik}$ and $\Omega_{ki}$ are defined as

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

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

Proof: See the proof of Theorem 1 in [9].

Theorem 5: (The non-adjacency condition) All non-adjacent pairs of jobs in an
optimal schedule must satisfy the following condition

$$\text{if } p_i = p_k, \text{ then } \beta_i(p_k + \Delta) - \Lambda_{ik}(\beta_i + \alpha_i) \geq \beta_k(p_i + \Delta) - \Lambda_{ki}(\beta_k + \alpha_k)$$

where job $i$ precedes job $k$, and $\Lambda_{ik}$ and $\Lambda_{ki}$ are defined as

$$\Lambda_{xy} = \begin{cases} 0 & \text{if } s_x \leq 0 \\ s_x & \text{if } 0 < s_x < p_y + \Delta \\ p_y + \Delta & \text{otherwise} \end{cases}$$
where $s_x = d_x - t - p_x$ is the slack of job $x$, $\Delta$ is the sum of the processing times of all jobs between $i$ and $k$, and $t$ is the sum of the processing times of all jobs preceding job $i$.

Proof: Similar to the proof of Theorem 1 in [9].

4. A HEURISTIC PROCEDURE

At the beginning of the branch-and-bound algorithm, we apply a heuristic procedure to compute an upper bound for the objective value of the problem. If the lower bound of a subproblem is greater than or equal to the upper bound, then this subproblem cannot yield a better solution, and hence we need not continue to branch from the corresponding node in the search tree. To stop the branching process in many nodes of the search tree, the upper bound should be as small as possible.

In this section, a two phase heuristic procedure is presented for minimizing the sum of weighted earliness and weighted tardiness. In the first phase we construct a schedule using a priority dispatching rule. The second phase consists of a local improvement procedure to improve the schedule obtained in the first phase. For the problem under consideration, an initial schedule can be produced by efficient priority dispatching rules. One of them, the EXP-ET (Exponential Earliness/Tardiness) rule by Ow and Morton[9] has been shown to perform well in most problem settings.

The EXP-ET rule uses the following priority index at any instant $t$ when the machine
is available:

\[
I_i(t) = \begin{cases} 
\frac{\beta_i}{p_i} & \text{if } s_i \leq 0 \\
\frac{\beta_i \exp\left[-\frac{(\alpha_i + \beta_i)s_i}{\alpha_i \bar{p}}\right]}{p_i} & \text{if } 0 < s_i \leq (\frac{\beta_i}{\alpha_i + \beta_i})k\bar{p} \\
\frac{\alpha_i^2(\frac{\beta_i}{p_i} - \frac{(\alpha_i + \beta_i)s_i}{kp_i\bar{p}})}{p_i} & \text{if } (\frac{\beta_i}{\alpha_i + \beta_i})k\bar{p} < s_i \leq k\bar{p} \\
\frac{-\alpha_i}{p_i} & \text{otherwise}
\end{cases}
\]

Where \( s_i = d_i - t - p_i \) is the slack of job \( i \) at time \( t \), \( \bar{p} \) is the average processing time of all remaining jobs, and \( k \) is a look-ahead parameter. The value of \( k \) is determined through experiments, and for a single machine, according to Vepsalainen and Morton[16], usually lies between 1 and 3. Every time the machine is available the indices of all remaining jobs are calculated and the job with the highest index is chosen to be processed next.

As pointed out by Ow and Morton[9], the EXP-ET rule reflects a priority that focuses on the tardiness cost of a job as its slack becomes small. On the other hand, when the slack is large, it is the earliness cost that dominates. The value of parameter \( k \) should reflect the average number of jobs that may clash in the future each time a sequencing decision is to be made. In general, when job due dates are close together, many jobs will clash and a large value of \( k \) should be used. On the other hand, in the case where due dates are evenly distributed, \( k \) should be small since few jobs will clash. In the first phase, the EXP-ET rule is used to generate an initial
schedule.

In the second phase we attempt to modify the schedule obtained in the first phase to get a better schedule. The modification is obtained by an insertion procedure followed by a swap procedure.

(1) Insertion procedure: Two jobs A and B are selected. Job A is inserted immediately before job B if it improves the objective value of the current schedule. The insertion procedure consists of $n$ iterations. At iteration $j, j = 1, 2, \ldots, n$, job A is selected as the job in position $j$ in the current schedule. A candidate for job B is selected from among $\lceil n/3 \rceil$ jobs nearest to job A, where $\lceil x \rceil$ is the greatest integer $\leq x$. The objective value of the schedule after inserting job A immediately before the candidate of job B is computed and compared with each one of the $\lceil n/3 \rceil$ cases. Job B is selected such that the resulting schedule has the minimum objective value. We restrict the candidate for job B because it seldom occurs that the best job B candidate is far away from the job A selected.

(2) Swap procedure: Two jobs A and B are selected. Jobs A and B are interchanged with all other jobs keep the same sequence if it improves the objective value of the current schedule. The swap procedure also consists of $n$ iterations. At iteration $j, j = 1, 2, \ldots, n$, job A is selected as the job in position $j$ in the current schedule. We select job B in the same way as in the insertion procedure.
5. IMPLEMENTATION OF THE BRANCH-AND-BOUND ALGORITHM

We now describe the implementation of our branch-and-bound algorithm.

Initially, we apply the heuristic procedure provided in section 4 to calculate an upper bound for the objective value of problem \((P)\). The upper bound is updated whenever a feasible schedule that improves the upper bound is generated during the branching process. Motivated by the results of Theorems 2 and 3, when the average tardiness factor of a problem is large \((\geq 0.5)\), we adopt the WSPT rule using \(\beta_i\) as the weight of job \(i\) for all \(i=1, 2,\ldots, n\) to sequence the jobs in the multiplier adjustment method, and we adopt a forward sequencing branching rule, where a node at level \(l\) of the search tree corresponds to a partial schedule with \(l\) jobs fixed in the first \(l\) positions. When the average tardiness factor of a problem is small \((\leq 0.5)\), we adopt the WLPT rule using \(\alpha_i\) as the weight of job \(i\) for all \(i=1, 2,\ldots, n\) to sequence the jobs in the multiplier adjustment method, and we adopt a backward sequencing branching rule, where a node at level \(l\) of the search tree corresponds to a partial schedule with \(l\) jobs fixed in the last \(l\) positions.

We use the depth-first strategy to search the tree, and break ties by selecting the node with the smallest cost of the associated partial schedule plus the associated lower bound. For each node in the search tree, we apply the following three tests to determine whether it should be discarded or not. In the first test, the dominance
rules presented in section 3 are applied. In the second test, the adjacent pairwise interchange technique is used to compare the cost of the two jobs most recently added to the partial schedule with the corresponding cost when these two jobs are interchanged. If the former cost is larger than the latter, the node is discarded.

Finally, if the node is not eliminated by the two tests above, a lower bound is calculated for this node. If the lower bound plus the cost of the associated partial schedule is larger than the current upper bound, the node is discarded.

6. COMPUTATIONAL EXPERIMENTS

The branch-and-bound algorithm is tested on problems with 15, 20, 30, 40 and 50 jobs that are generated as follows. For each job i, an integer processing time $p_i$, an integer earliness weight $\alpha_i$, and an integer tardiness weight $\beta_i$ are generated from the uniform distribution $[1, 10]$. For each job i, an integer due date $d_i$ is generated from the uniform distribution $[P(1-T-R/2), P(1-T+R/2)]$ where $P$ is the total processing time, $T$ is the average tardiness factor, set at 0.0, 0.2, 0.6, 0.8 and 1.0, and $R$ is the relative range of due dates, set at 0.2, 0.4, 0.6 and 0.8. For each combination of $n$, $T$ and $R$, 5 problems are generated, yielding 100 problems for each value of $n$.

The algorithm is coded in C and implemented on a Pentium II-266 personal computer. To prevent excessive computation time, whenever a problem is not solved
within the time limit of 3600 seconds (one hour), computation is stopped for that problem.

To test the effectiveness of the lower bound proposed in this paper, we compare it with the lower bound developed by Li[10]. Since both lower bounds can be computed in $O(n)$ time, the effectiveness is measured by the average ratio of (lower bound proposed)/(Li’s lower bound). The comparative results are given in Table 1. It is clear that for the set of problems generated our lower bound is superior to Li’s lower bound for all problem sizes.

To evaluate the performances of the lower and upper bounds used, the average percentage deviations from the optimum ($Z$) of the initial lower and upper bounds are measured and reported in Table 2. As can be seen from this table, both the lower and upper bounds are very tight. The initial lower bounds deviate from the optimum by 9.08% on average, and the initial upper bounds deviate from the optimum by 2.77% on average.

<table>
<thead>
<tr>
<th>$n$</th>
<th>Lower bound proposed</th>
<th>Li’s lower bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>1.110</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1.090</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1.106</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>1.087</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>1.057</td>
<td></td>
</tr>
<tr>
<td><strong>average</strong></td>
<td><strong>1.090</strong></td>
<td><strong>1.090</strong></td>
</tr>
</tbody>
</table>
Table 2  Performances of the initial lower bound (LB) and upper bound (UB).

<table>
<thead>
<tr>
<th>n</th>
<th>$100 \times (Z - LB)/Z$</th>
<th>$100 \times (UB - Z)/Z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>11.95</td>
<td>3.20</td>
</tr>
<tr>
<td>20</td>
<td>10.45</td>
<td>3.75</td>
</tr>
<tr>
<td>30</td>
<td>11.10</td>
<td>3.70</td>
</tr>
<tr>
<td>40</td>
<td>8.20</td>
<td>2.25</td>
</tr>
<tr>
<td>50</td>
<td>3.70</td>
<td>0.95</td>
</tr>
<tr>
<td>average</td>
<td>9.08</td>
<td>2.77</td>
</tr>
</tbody>
</table>

\(^a\)The average percentage of 90 problems since 10 problems are not solved in one hour.

\(^b\)The average percentage of 63 problems since 37 problems are not solved in one hour.

To evaluate the effectiveness of the node-fathoming tests, the average percentages of nodes fathomed (number of nodes fathomed/total number of nodes fathomed) by each of these tests are measured and reported in Table 3. In Table 3, Test1 represents the application of the dominance rules given in section 3, Test2 represents the application of the adjacent pairwise interchange technique, and Test3 represents the comparison of the lower bound plus the cost of the associated partial schedule to the current upper bound. Table 3 shows that the average percentage of nodes fathomed by Test1 increases as the number of jobs increases. This is due to the fact that the larger the number of jobs the more likely two jobs will have the same amount of processing time, and hence the more likely the non-adjacency condition in Theorem 5 will hold. In general, Test3 is most effective at fathoming nodes when the number of jobs is small ($n \leq 20$), while Test1 becomes more effective at fathoming nodes as the number of jobs increases. The overall average percentage of nodes fathomed is 43.59%, 14.46% and 41.95% for Test1, Test2 and Test3, respectively.
Table 3  Effectiveness of the node-fathoming tests.

<table>
<thead>
<tr>
<th>N</th>
<th>Test1</th>
<th>Test2</th>
<th>Test3</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>23.15</td>
<td>15.42</td>
<td>61.43</td>
</tr>
<tr>
<td>20</td>
<td>30.35</td>
<td>12.27</td>
<td>57.38</td>
</tr>
<tr>
<td>30</td>
<td>54.34</td>
<td>16.48</td>
<td>29.14</td>
</tr>
<tr>
<td>40</td>
<td>54.51</td>
<td>13.89</td>
<td>31.60</td>
</tr>
<tr>
<td>50</td>
<td>55.62</td>
<td>14.22</td>
<td>30.16</td>
</tr>
</tbody>
</table>

Average 43.59 14.46 41.95

*The average percentage of 100 problems for each value of n.*

Table 4 presents the aggregated computational results for the branch-and-bound algorithm. The results include the average CPU time in seconds, the average number of nodes generated, and the number of problems unsolved (out of 100).

Note that the values of average CPU time and average number of nodes generated in Table 4 do not include consideration of the abandoned problems. Table 4 shows that for n=40, 10 problems are unsolved within 3600 seconds. This number increases to 37 for n=50. The average CPU time for n=40 is 316.06 seconds, while for n=50 it rises to 381.86 seconds.

Table 4  Performance of the branch-and-bound algorithm.

<table>
<thead>
<tr>
<th>n</th>
<th>Average CPU time (seconds)</th>
<th>Average number of nodes</th>
<th>Number of problems unsolved (out of 100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0.05</td>
<td>102</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0.31</td>
<td>362</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>34.98</td>
<td>30,936</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>316.06</td>
<td>203,212</td>
<td>10</td>
</tr>
<tr>
<td>50</td>
<td>381.86</td>
<td>114,252</td>
<td>37</td>
</tr>
</tbody>
</table>
Finally, the influence of the average tardiness factor $T$ and the relative range of due dates $R$ on problem hardness is analyzed. Table 5 gives the average CPU time, the average number of nodes generated, and the number of problems unsolved (out of 5) for each pair of values of $T$ and $R$ for $n=40$ and 50. Results for other problem sizes are similar. Note that due to the symmetry of the problem, problems with average tardiness factors $T$ and $1-T$ are likely to be of similar difficulty. It can be seen from Table 5 the problems are most difficult when $T=0.6$. This is expected since the results of Theorems 2 and 3 imply that our lower bound can be very tight when most jobs are tardy or early. Note that the influence of $R$ is not very significant when $T=0.6$. However, increasing $R$ seems to make the problems harder for other values of $T$.

Table 5 Influence of the tardiness factor ($T$) and the due date range ($R$) for $n=40$ and 50.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>0.0</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td>$R$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n=40$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>8.0: 2,671: 0</td>
<td>81.8: 29,686: 0</td>
<td>24.8: 8,623: 0</td>
<td>105.6: 41,952: 0</td>
</tr>
<tr>
<td>0.2</td>
<td>14.6: 7,245: 0</td>
<td>26.2: 11,409: 0</td>
<td>245.6: 133,532: 0</td>
<td>1576.5: 712,637: 1</td>
</tr>
<tr>
<td>0.6</td>
<td>2877.0: 2,937,680: 3</td>
<td>906.0: 910,756: 1</td>
<td>739.3: 429,486: 2</td>
<td>1331.5: 611,937: 3</td>
</tr>
<tr>
<td></td>
<td>52.2: 27,797: 0</td>
<td>337.2: 214,321: 0</td>
<td>447.2: 200,991: 0</td>
<td>515.6: 231,407: 0</td>
</tr>
<tr>
<td>1.0</td>
<td>0.6: 140: 0</td>
<td>7.0: 1,643: 0</td>
<td>25.6: 9,764: 0</td>
<td>171.0: 65,156: 0</td>
</tr>
<tr>
<td>$n=50$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>291.0: 58,670: 0</td>
<td>81.6: 17,982: 0</td>
<td>398.8: 94,609: 0</td>
<td>710.0: 199,794: 3</td>
</tr>
<tr>
<td>0.2</td>
<td>695.0: 287,850: 0</td>
<td>622.0: 127,598: 0</td>
<td>264.5: 70,797: 3</td>
<td>3508.0: 913,703: 4</td>
</tr>
<tr>
<td>0.6</td>
<td>---: ---: 5</td>
<td>---: ---: 5</td>
<td>2399.7: 982,543: 2</td>
<td>608.0: 203,415: 4</td>
</tr>
<tr>
<td>0.8</td>
<td>53.0: 30,416: 3</td>
<td>267.5: 61,029: 1</td>
<td>1013.7: 304,984: 1</td>
<td>2820.0: 672,904: 4</td>
</tr>
</tbody>
</table>
7. CONCLUSIONS

In this paper, the problem of scheduling a given set of independent jobs on a single machine to minimize the sum of weighted earliness and weighted tardiness without considering machine idle time is studied. A branch-and-bound algorithm based on powerful lower and upper bounding procedures is presented. Simple (conditional) dominance rules are also derived to help eliminating unpromising nodes in the branch-and-bound search tree. Computational results show that the branch-and-bound algorithm performs very well on problems with up to 50 jobs.

One way in which the performance of the algorithm might be improved is to use tighter but computationally slower lower and upper bounds. As pointed out by Potts and van Wassenhove[22], the use of tighter but slower bounds within a conventional branch-and-bound procedure has so far not proven to be effective. However, more efficient bounding procedures that can generate tighter bounds than ours but require little extra computational effort will be necessary for solving larger sized problems. Also, additional research to develop stronger (unconditional) dominance rules to further cut down the size of the search tree may permit much larger sized problems to be solved.
REFERENCES


8.  Abdul-Razaq, T. and Potts, C.N., Dynamic programming state-space relaxation
    39(2), 141-152.


10. Li, G., Single machine earliness and tardiness scheduling. *European Journal of

11. Szwarc, W., Adjacent orderings in single-machine scheduling with earliness and

    involving total tardiness and total earliness penalties. *International Journal of

    tardiness and earliness scheduling problems. *European Journal of Operational


Why this paper is important?

With the growing interest in Just-In-Time (JIT) production philosophy, recent research on single machine scheduling problems has focused on the performance measures involving both job earliness and tardiness costs. In this paper, we address a single machine scheduling problem where the objective is to minimize the sum of weighted earliness cost and weighted tardiness cost, subject to the constraint that no machine idle time is allowed. The assumption of no machine idle time represents a type of production setting where the machine idle cost is larger than the job earliness cost, or the machine capacity is limited compared to demand. The problem is known to be NP-hard. We present an efficient branch-and-bound algorithm. Computational experiments show that the algorithm performs very well.