INFORMS Journal on Computing
Vol. 8, No. 4, Fall 1996

0899-1499/96/0804-0402 $01 25
© 1996 INFORMS

A Branch-and-Bound Algorithm for Single-Machine Earliness–Tardiness Scheduling with Idle Time

J. A. HOOGEVEEN / *Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands, Email address: slam@win.tue.nl*

S. L. VAN DE VELDE / *Department of Mechanical Engineering, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands, Email address: s.l.vandevelde@wb.utwente.nl*

(Received: February 1992; revised: September 1995; accepted: October 1995)

We address the NP-hard single-machine problem of scheduling n independent jobs so as to minimize the sum of α times total completion time and β times total earliness with $\beta > \alpha$, which can be rewritten as an earliness–tardiness problem. Postponing jobs by leaving the machine idle may then be advantageous. The allowance of machine idle time between the execution of jobs singles out our problem from most concurrent research on problems with earliness penalties. Solving the problem to optimality poses a computational challenge, since the possibility of leaving the machine idle has a major effect on designing a branch-and-bound algorithm in general, and on computing lower bounds in particular. We present a branch-and-bound algorithm which is based upon many dominance rules and various lower bound approaches, including relaxation of the machine capacity, data manipulation, and Lagrangian relaxation. The algorithm is shown to solve small instances with up to 20 jobs.

Recently, we have seen much interest in machine scheduling models that penalize both early and tardy completions of jobs. We refer to Baker and Scudder[1] for an overview. These types of models are supposed to capture the just-in-time concept, whose basic premise is to reduce costly inventories by enforcing on-time deliveries throughout the entire manufacturing process. In theory, on-time deliveries may be achieved by allowing machine idle time. In practice, however, idle time is highly controversial, because of lost production capacity, for instance.

In this paper, we consider an NP-hard single-machine earliness–tardiness problem in which the insertion of machine idle time is allowed, and present a branch-and-bound algorithm for its solution. As we will see later, the possibility to leave the machine idle poses a computational challenge and affects significantly the design of a branch-and-bound algorithm.

We consider the following problem. A set $\mathcal{J} = \{J_1, \ldots, J_n\}$ of n independent jobs has to be scheduled on a single machine that is continuously available from time zero onward. The machine can handle at most one job at a time. Job J_j ($j = 1, \ldots, n$) requires a positive integral uninterrupted processing time p_j and should ideally be completed exactly on its due date d_j. A *schedule* specifies for each job J_j a completion time C_j such that the jobs do not overlap in their execution. The order in which the machine processes the jobs is called the *job sequence*. For a given schedule, the earliness of J_j is defined as $E_j = \max\{0, d_j - C_j\}$ and its tardiness as $T_j = \max\{0, C_j - d_j\}$. In addition, we define *maximum earliness* as $E_{\max} = \max_{1 \leq j \leq n} E_j$ and *maximum tardiness* as $T_{\max} = \max_{1 \leq j \leq n} T_j$. Accordingly, J_j is called *early, just-in-time,* or *tardy* if $C_j < d_j$, $C_j = d_j$, or $C_j > d_j$, respectively. The cost of a schedule σ is the weighted sum of total completion time and total earliness, that is,

$$f(\sigma) = \alpha \sum_{j=1}^{n} C_j + \beta \sum_{j=1}^{n} E_j,$$

where α and β are given positive weights with $\beta > \alpha$. Without loss of generality, we also assume α and β to be integral and relatively prime. We are interested in the case $\beta > \alpha$, since insertion of machine idle time may be advantageous only in this case. The cost function $f(\sigma)$ arguably measures inventory costs in a machine scheduling environment: total completion time measures the work-in-process inventories, and total earliness measures the storage inventories due to early completions. The problem, referred to as problem (P), is to find a feasible schedule σ that minimizes $f(\sigma)$.

Problem (P) is NP-hard. By definition we have that $T_j = C_j + E_j - d_j$ for $j = 1, \ldots, n$, and the cost function can therefore alternatively be written as

$$(\alpha - \gamma) \sum_{j=1}^{n} C_j + (\beta - \alpha) \sum_{j=1}^{n} E_j + \gamma \sum_{j=1}^{n} T_j + \gamma \sum_{j=1}^{n} d_j,$$

for any $0 \leq \gamma \leq \alpha$. Garey, Tarjan, and Wilfong[5] prove that minimizing this cost function with $\gamma = \alpha$ and $\beta > \alpha$ is NP-hard.

Problem (P) was identified by Kanet and Christie[12] and studied by Fry and Leong.[3] They formulated it as an integer linear problem and used a standard code to find an optimal schedule. Not surprisingly, this method already requires excessive computation times for small instances.

Subject classifications: Single-machine scheduling.
Other key words. Earliness, tardiness, machine idle-time, branch and bound algorithm, Lagrangian relaxation.

Their formulation is 'weak' in that the linear programming relaxation gives weak lower bounds, which seriously impairs the performance of any standard integer linear program solver. Also, a general code does not take advantage of the problem structure. Fry, Leong, and Rakes[4] compare the performance of the integer linear programming approach with the performance of a rudimentary branch-and-bound algorithm for the problem $1\|\Sigma_{j=1}^{n}(\alpha C_j + \beta E_j + \gamma T_j)$ in which idle time is allowed; indeed, they find that their branch-and-bound algorithm is much faster. Inspecting the alternative rendition of the cost function we see that their problem is equivalent with ours.

Both our algorithm and the branch-and-bound algorithm by Fry, Leong, and Rakes hinge upon the observation that the search for an optimal schedule can be reduced to a search over the $n!$ different job sequences. This is possible, since there is a clear-cut method to insert machine idle time to minimize total cost for any *given* sequence. This method, which requires $O(n^2)$ time, is described in Section 1. In Section 2, we discuss the design of the branch-and-bound algorithm, including the upper bound, the branching rule, the search strategy, and the dominance rules. The derivation of lower bounds is significantly complicated by the possibility of machine idle time. The range of the due dates in proportion to the processing times mainly determines how much idle time is desired. To cope with the different problem instances, we present five approaches for lower bound computation, including Lagrangian relaxation, in Section 3. The branch-and-bound algorithm is based upon many dominance rules and various lower bound approaches. Unfortunately, it can handle only small problem sizes; the computational results presented in Section 4 exhibit that we can solve instances with up to 20 jobs. Conclusions are given in Section 5.

Throughout the paper, we follow the three-field notation of Graham, Lawler, Lenstra, and Rinnooy Kan[6] to classify scheduling problems.

1. The Insertion of Idle Time for a Given Sequence

In this section, we describe a procedure to insert machine idle time so as to minimize total cost for a *given* sequence. This procedure is not new. Similar methods have been presented (cf. Baker and Scudder[11]), including those presented by Fry, Leong, and Rakes[4] for the $1\|\Sigma_{j=1}^{n}(\alpha C_j + \beta E_j + \gamma T_j)$ problem and by Garey, Tarjan, and Wilfong[5] for the $1\|\Sigma_{j=1}^{n}(E_j + T_j)$ problem. This is not surprising: as we have already noted, $T_j = C_j + E_j - d_j$ for all j; for specific choices for α and β, our problem is equivalent with theirs. Since the basis of the procedure is well known, we just present our implementation. For a proof of correctness, see Hoogeveen and van de Velde[10].

Suppose that the scheduling order is $\sigma = (J_n, \ldots, J_1)$. Since no job can start before time zero, we need the constraint that $C_l \geq \Sigma_{j=l}^{n} p_j$ for $l = 1, \ldots, n$. We use an inductive procedure for finding an optimal schedule for σ. It finds an optimal schedule for the subsequence $(J_l, \ldots, J_1)$, given an optimal schedule for the subsequence $(J_{l-1}, \ldots, J_1)$, for $l = 2, \ldots, n$; we initiate the procedure by scheduling J_1 to be completed at time $\max\{d_1, \Sigma_{j=1}^{n} p_j\}$. When adding J_l to the subsequence $(J_{l-1}, \ldots, J_1)$, we first check whether $\Sigma_{j=l}^{n} p_j \leq d_l \leq C_{l-1} - p_{l-1}$; if so, then putting $C_l = d_l$ yields an optimal schedule for $(J_l, \ldots, J_1)$. If $d_l < \Sigma_{j=l}^{n} p_l$, then we obtain an optimal schedule for $(J_l, \ldots, J_n)$ by putting $C_l = \Sigma_{j=l}^{n} p_j$. If $d_l > C_{l-1} - p_{l-1}$, then we tentatively put $C_l = C_{l-1} - p_{l-1}$. Define $\mathcal{Q}_l$ as the set containing J_l and its immediate followers, that is, the group of jobs that are executed after J_l with no machine idle time in between. We now compute the optimal delay of the jobs in $\mathcal{Q}_l$, disregarding the jobs not in $\mathcal{Q}_l$. A delay of one unit of time increases the completion time of each job in $\mathcal{Q}_l$ by one and decreases the earliness of each early job in $\mathcal{Q}_l$ by one; the total effect on the cost is equal to the primitive directional derivative $g_l = \alpha|\mathcal{Q}_l| - \beta n_l$, where $|\mathcal{Q}_l|$ denotes the number of jobs in $\mathcal{Q}_l$ and n_l denotes the number of early jobs in $\mathcal{Q}_l$. Obviously, we defer the jobs in $\mathcal{Q}_l$ until g_l becomes nonnegative, that is, as long as $n_l \leq \alpha|\mathcal{Q}_l|/\beta$. Define $a = \lfloor \alpha|\mathcal{Q}_l|/\beta \rfloor$, $k = |\mathcal{Q}_l| - a$, and $E_{[k]}$ as the kth smallest value of the earliness of the jobs in $\mathcal{Q}_l$. If the jobs in $\mathcal{Q}_l$ are deferred by $\delta = E_{[k]}$, then at most a jobs in $\mathcal{Q}_l$ remain early and, due to the choice of a, g_l then becomes nonnegative. Deferring the jobs by δ is only feasible if δ is no larger than the length $\delta_{\max}$ of the period of idle time immediately after the last job in $\mathcal{Q}_l$. If $\delta \leq \delta_{\max}$, then we get an optimal schedule for $(J_l, \ldots, J_1)$ by deferring the jobs in $\mathcal{Q}_l$ by δ. If $\delta > \delta_{\max}$, then we defer the jobs in $\mathcal{Q}_l$ by $\delta_{\max}$. At this point, we repeat the process for J_l: we update $\mathcal{Q}_l$, and evaluate if additional delay of the jobs in $\mathcal{Q}_l$ is advantageous. We now give a step-wise description of the idle time insertion algorithm.

Idle Time Insertion Algorithm

```
Step 1. $C_1 \leftarrow \max\{d_1, \Sigma_{j=1}^{n} p_j\}$; $l \leftarrow 2$.
Step 2. If $l = n + 1$, then go to Step 10.
Step 3. If $d_l < \Sigma_{j=1}^{n} p_j$, then $C_l \leftarrow \Sigma_{j=l}^{n} p_j$; otherwise, put $C_l \leftarrow$
    $\min\{d_l, C_{l-1} - p_{l-1}\}$. If $C_l = d_l$, then go to Step 9.
Step 4. Determine $\mathcal{Q}_l$ and evaluate $g_l$. If $g_l \geq 0$, then go to
    Step 9.
Step 5. Compute $E_j$ for each job $J_j \in \mathcal{Q}_l$.
Step 6. Compute $\delta_{\max}$, i.e., the length of the period of idle
    time immediately after the last job in $\mathcal{Q}_l$.
Step 7. Let $a \leftarrow \lfloor |\mathcal{Q}_l|\alpha/\beta \rfloor$, and $k \leftarrow |\mathcal{Q}_l| - a$. Determine $\delta$ as
    the $k$th smallest earliness value for the jobs in $\mathcal{Q}_l$.
Step 8. Defer the jobs in $\mathcal{Q}_l$ by $\min\{\delta, \delta_{\max}\}$. If $\delta > \delta_{\max}$, then
    go to Step 4.
Step 9. $l \leftarrow l + 1$; go to Step 2.
Step 10. Stop: an optimal schedule for the sequence $(J_n, \ldots, J_1)$
    has been determined.
```

Theorem 1: *The idle time insertion algorithm generates an optimal schedule for a given sequence.*

For a proof, see [10]. As to the complexity of the algorithm, we have that a complete run through the main part of the algorithm, i.e., Steps 4 through 8, takes $O(n)$ time: this is needed to identify the set $\mathcal{Q}_l$, to compute the primitive directional derivative g_l, the values $\delta_{\max}$ and δ, and to defer the jobs, if necessary. The value δ is determined in $O(n)$ time through a median-finding technique. After each run through the main part of the algorithm, a gap between two successive jobs is closed. As at most $n - 2$ such gaps exist, the algorithm runs in $O(n^2)$ time. For the case $2\alpha = \beta$, i.e., for the problem $1\|\Sigma_{j=1}^{n}(E_j + T_j)$, Garey, Tarjan, and Wilfong[5] show that the idle time insertion procedure can be implemented to run in $O(n \log n)$ time. The problem of inserting machine idle time can also be solved by a symmetric procedure starting with the first job in σ. Because of our specific branching rule, however, we choose to start at the end.

In the remainder, we use the terms *sequence* and *schedule* interchangeably. Unless stated otherwise, σ also refers to the optimal schedule for the sequence σ and to the set of jobs in the sequence σ. From now on, we let $p(\sigma) = \Sigma_{J_j \in \sigma} p_j$.

2. Dominance Criteria

We adopt a *backward sequencing branching rule*: each node at level k of the search tree corresponds to a sequence π with k jobs fixed in the last k positions. At level k, there are $n - k$ descendant nodes: one for each unscheduled job. By branching from a node corresponding to some π, we add some unscheduled job J_j to π, thereby obtaining the sequence $J_j\pi$. We explore the tree by a *depth-first* strategy, and we employ an *active node* search: at each level we choose one node to branch from, and we consistently choose an unfathomed node whose job has the largest due date.

The completion times of the jobs in π are only temporary: when we add some J_j to π, we determine the optimal schedule for $J_j\pi$ by possibly deferring some jobs in π. Before entering the search tree, we determine an upper bound on the optimal solution value; it is obtained by sequencing the jobs in order of nondecreasing $d_j - p_j$ to get an initial solution, and by then trying to reduce its cost by swapping adjacent jobs.

Let $f(\pi)$ denote the minimal cost for π. Let $\bar{f}(\pi)$ denote the minimal cost for π if the first job may be started before time $p(\mathcal{J} - \pi)$. For any partial schedule π, we have $f(\pi) \geq \bar{f}(\pi)$. A node is discarded if its associated partial schedule π cannot lead to a complete schedule with cost less than the incumbent upper bound value UB. Let $\text{LB}(\mathcal{J} - \pi)$ be some lower bound on the minimal cost of scheduling the jobs in the set $\mathcal{J} - \pi$. Obviously, we discard a node if $f(\pi) + \text{LB}(\mathcal{J} - \pi) \geq \text{UB}$. The following rules are due to our backward sequencing branching strategy. Let $g(\sigma_1, \sigma_2)$ be a lower bound on the cost for scheduling the jobs in σ_1 given the final partial schedule σ_2.

Theorem 2: *The partial schedule π can be discarded if there exists a $J_j \in \mathcal{J} - \pi$ for which $\bar{f}(J_j\pi) + g(\mathcal{J} - \pi - J_j, \pi) \geq \text{UB}$.*

Proof: Consider a complete sequence σ that has π as final subsequence. Thus, σ can be written as $\sigma = \pi_1 J_j \pi_2 \pi$. Accordingly, we have

$$f(\sigma) = f(\pi_1 J_j \pi_2 \pi) \geq \bar{f}(J_j\pi) + g(\pi_1\pi_2, \pi) \geq \text{UB}. \quad \blacksquare$$

It is essential that $g(\mathcal{J} - \pi - J_j, \pi)$ depends only on π and not on $J_j\pi$, and that we use $\bar{f}(J_j\pi)$ instead of $f(J_j\pi)$. We derive two corollaries from Theorem 2.

Corollary 1: *If for a given partial schedule π we have that $\bar{f}(J_jJ_k\pi) + g(\mathcal{J} - \pi - J_j - J_k, \pi) \geq \text{UB}$ for some $J_j \in \mathcal{J} - \pi$ and $J_k \in \mathcal{J} - \pi$, then J_k precedes J_j in any complete schedule $\sigma\pi$ with $f(\sigma\pi) < \text{UB}$.*

Corollary 2: *The partial schedule π can be discarded if two jobs $J_j \in \mathcal{J} - \pi$ and $J_k \in \mathcal{J} - \pi$ exist with $g(\mathcal{J} - \pi - J_j - J_k, \pi) + \min\{\bar{f}(J_jJ_k\pi), \bar{f}(J_kJ_j\pi)\} \leq \text{UB}$.*

If a partial schedule $\pi^* \neq \pi$ exists comprising the same jobs as π and having $f(\sigma\pi^*) \leq f(\sigma\pi)$ for any sequence σ for the remaining jobs, then we can also discard π. If $f(\sigma\pi^*) < f(\sigma\pi)$ for some σ, then π is *dominated* by π^*. If $f(\sigma\pi^*) = f(\sigma\pi)$ for every σ, then we discard either π^* or π. The dominance condition above can be narrowed by the requirement that $f(\pi^*) \leq f(\pi)$ and that the circumstances to add the remaining jobs to π^* are at least as good as the circumstances to add the remaining jobs to π. The question whether such a sequence π^* exists is of course NP-complete. We strive therefore to identify sufficient conditions to discard π. The temporary nature of the job completion times for π complicates the achievement of this goal. We have to be careful with dominance conditions that are based on interchange arguments: the conditions must remain valid if the jobs in π are deferred.

Suppose that the jobs in π have been reindexed in order of increasing completion times. In each of the following theorems, stating the dominance rules, the sequence π^* is obtained from π by swapping two jobs, say, J_j and J_k. We do not compute the optimal completion times for the sequence π^*. Instead, we determine the job completion times for the sequence π^* as follows. Let C_i and C_i^* denote the completion time of J_i in the schedule π and π^*, respectively. Then we let

$$\begin{aligned}
C_i^* &= C_i, && \text{for } i = 1, \ldots, j-1, i = k+1, \ldots, |\pi|,\\
C_i^* &= C_i - p_j + p_k, && \text{for } i = j+1, \ldots, k-1,\\
C_k^* &= C_j - p_j + p_k,\\
C_j^* &= C_k.
\end{aligned}$$

Let $F(\pi^*)$ be the cost associated with the completion times C_i^*, for $i = 1, \ldots, |\pi|$. Hence, $F(\pi^*) \geq f(\pi^*)$. To validate the following dominance rules, we must verify that $f(\pi) \geq F(\pi^*)$, even if the jobs are deferred. Due to the relation between π and π^*, this comes down to verifying that for each set of nonnegative values Δ_i $(i = 1, \ldots, n)$ we have

$$\alpha \sum_{i=j}^{k} C_i + \beta \sum_{i=j}^{k} \max\{0, d_i - C_i - \Delta_i\}$$
$$\geq \alpha \sum_{i=j}^{k} C_i^* + \beta \sum_{i=j}^{k} \max\{0, d_i - C_i^* - \Delta_i\}. \quad (1)$$

We derive a number of sufficient conditions under which (1) holds; some are obvious, some are quite involved. The proofs proceed by interchange arguments and are found in Hoogeveen and van de Velde.[10]

Theorem 3: *There is an optimal schedule with J_j preceding J_k if $p_j = p_k$ and $d_j \leq d_k$.*

Theorem 4: *The partial sequence π can be discarded if there are two jobs J_j and J_k with $C_k = C_j + \Sigma_{i=j+1}^{k} p_i$ and $p_j > p_k$ for which*

$$\alpha \sum_{i=j}^{k} C_i + \beta \sum_{i=j+1}^{k} \max\{0, d_i - C_i\}$$
$$\geq \alpha \sum_{i=j}^{k} C_i^* + \beta \sum_{i=j+1}^{k} \max\{0, d_i - C_i^*\}.$$

Note that the possible increase of E_j is excluded from the condition in Theorem 4. The following theorem shows that if no idle time exists between two adjacent jobs, then we only have to check whether condition (1) holds for the current set of completion times to obtain dominance.

Theorem 5: *The partial sequence π can be discarded if there are two jobs J_j and J_k with $C_k = C_j + p_k$ and $p_j > p_k$ for which*

$$\alpha(p_j - p_k) + \beta \max\{0, d_j - C_j\} + \beta \max\{0, d_k - C_k\} \geq \beta \max\{0, d_j - C_k\} + \beta \max\{0, d_k - C_k + p_j\}. \quad (2)$$

In Corollary 3, explicit conditions for the existence of dominance are derived from Theorem 5. We use this corollary in our derivation of lower bounds in Section 3.

Corollary 3: *The partial sequence π can be discarded if there are two adjacent jobs J_j and J_k with $C_k = C_j + p_k$ and $p_j > p_k$ that satisfy one of the following conditions:*

$$C_k - p_j \geq d_k,$$

$$C_k - p_j < d_k,\ C_k \geq d_j$$
$$C_j \geq d_j, \text{ and } \alpha(p_j - p_k) \geq \beta(d_k - C_k + p_j),$$

$$C_k - p_j < d_k,$$
$$C_k < d_k,\quad C_j \geq d_j, \text{ and } \alpha(p_j - p_k) \geq \beta p_j,$$

$$C_k - p_j < d_k,\ C_k \geq d_k,$$
$$C_j < d_j, \text{ and } \alpha(p_j - p_k) \geq \beta(d_k - d_j - p_k + p_j).$$

Theorem 6: *The partial sequence π with J_k scheduled last is dominated if there is a J_j such that $p_j > p_k$ and $C_j - p_j + p_k \geq d_k$.*

Theorem 7: *There is an optimal schedule in which J_k is not scheduled in the last position, if there is some J_j with $p_j > p_k$ and $d_j - p_j \geq d_k - p_k$.*

Corollary 4: *There is an optimal schedule in which J_j is scheduled last if $p_j \geq p_k$ and $d_j - p_j \geq d_k - p_k$ for each $J_k \in \mathcal{J}$.*

3. Lower Bounds

We present five lower bound procedures, as it seems to be impossible to develop a lower bound procedure that copes satisfactorily with all conceivable due date patterns. Each procedure is effective for a specific class of instances. Nonetheless, we use them in a supplementary rather than a complementary fashion. We partition the job set $\mathcal{J}$ into subsets, apply each lower bound method to each subset, and aggregate the best lower bounds. The success of this approach depends on the partitioning strategy. The jobs in a subset should be conflicting, that is, they should overlap when completed at their due date. If they are not, then we get the weak lower bound $\alpha \sum_{j=1}^{n} d_j$. In this sense, we prefer subsets such that the executions of the jobs in the same subset interfere with each other, but not with the execution of the jobs in the other subsets. We propose two partitioning strategies that pursue this effect.

The first strategy is motivated by the structure of any optimal schedule. The jobs that are consecutively processed between two periods of idle time interfere with each other, but not with the other jobs. Such a partitioning is hard to obtain. To mimic such a partitioning, we specify clusters. A *cluster* is a set of jobs such that for each job J_j in the cluster there is another job J_k in the cluster such that the intervals $[d_j - p_j, d_j]$ and $[d_k - p_k, d_k]$ overlap; hence, for each job in the cluster there exists a conflict with at least one other job in the cluster. However, clusters may interfere with each other in any optimal schedule. Intuitively, we like to have as few clusters as possible. We try to achieve this by a simple greedy algorithm. In fact, if there is only one cluster, then we can solve the problem in either $O(n^2)$ or $O(n^2 \sum_{j=1}^{n} p_j)$ time, depending on whether the $d_j - p_j$ are large or small relative to the sum of the processing times (Hoogeveen and van de Velde[11]).

The second strategy is the following. Given a partial schedule π, we try to identify the jobs not in π that will be early in any optimal complete schedule of the form $\sigma\pi$. We call these jobs *surely early*. The idea is to derive an upper bound T on the completion times of the unscheduled jobs; accordingly, $J_j \in \mathcal{J} - \pi$ is surely early if $d_j > T$. For instance, let g be the primitive directional derivative for deferring the first job in π by one unit. If we have that $|\mathcal{J} - \pi|(\beta - \alpha) \leq g$, then we know that the current set of completion times for the jobs in π is optimal for any schedule $\sigma\pi$; an upper bound T is then the start time of the first job in π. Other upper bounds are derived from the dominance rules. Suppose J_j and J_k are adjacent in π with $p_j > p_k$ and J_j preceding J_k. The first condition of Corollary 3 indicates that π is dominated if $C_k \geq d_k + p_j$; hence, an upper bound is given by $d_k + p_j - 1 - \sum_{J_i \in \pi,\, C_i \leq C_k} p_i$. From the other criteria in Corollary 3 and from Theorem 7, similar upper bounds are derived. They can also be derived from Theorem 4, but this requires an intricate procedure. Finally, we set T equal to the minimum of all upper bounds. If no upper bound is specified, then we let $T = \max_{1 \leq j \leq n} d_j + \sum_{j=1}^{n} p_j$.

3.1 Lower Bound 1: Relax the Objective Function

Let $\mathscr{E}$ denote the set of surely early jobs; let $\mathscr{R}$ be the set of remaining jobs. Observe that

$$\min_{\sigma \in \Omega} f(\sigma) \geq \min_{\sigma \in \Omega_{\mathscr{R}}} \sum_{J_j \in \mathscr{R}} \alpha C_j + \min_{\sigma \in \Omega_{\mathscr{E}}} \sum_{J_j \in \mathscr{E}} [\alpha C_j + \beta E_j],$$

where $\Omega_{\mathscr{R}}$ and $\Omega_{\mathscr{E}}$ denote the set of feasible schedules for the jobs in $\mathscr{R}$ and $\mathscr{E}$. The problem of minimizing $\sum_{J_j \in \mathscr{E}} [\alpha C_j + \beta E_j]$ is solvable in polynomial time: we have $E_j = d_j - C_j$ for each $J_j \in \mathscr{E}$, and hence, the scheduling cost comes down to $\sum_{J_j \in \mathscr{E}} [(\alpha - \beta)C_j + \beta d_j]$. Applying an analog of Smith's rule (Smith[14]), we minimize this cost component by scheduling the jobs in $\mathscr{E}$ in the interval $[T - p(\mathscr{E}), T]$ in order of nonincreasing processing times; the correctness of this rule is easily verified by an interchange argument. The other subproblem is solved by Smith's rule: simply schedule the jobs in $\mathscr{R}$ in nondecreasing order of their processing times in the interval $[0, p(\mathscr{R})]$.

A slight improvement of the lower bound is possible. Let $E^*_{\max}$ be the minimum maximum earliness for the jobs in $\mathscr{R}$ if they are processed in the interval $[0, p(\mathscr{R})]$. We compute $E^*_{\max}$ from the minimum-slack-time sequence, that is, the sequence in which the jobs appear in order of nondecreasing values $d_j - p_j$. Avoiding $E^*_{\max}$ requires at least $E^*_{\max}$ units of machine idle time. The lower bound can therefore be improved by $\alpha E^*_{\max}$. This lower bound approach can only be applied in conjunction with Theorem 2 if $\mathscr{E} = \emptyset$.

Since all jobs in $\mathscr{R}$ are scheduled in the interval $[0, p(\mathscr{R})]$, and since only one early job in $\mathscr{R}$ is taken into account, this lower bound is only effective if the due dates are small relative to the sum of the processing times.

3.2 Lower Bound 2: Relax the Machine Capacity

Recall that we can rewrite the objective function alternatively as $f(\sigma) = (\beta - \alpha) \sum_{j=1}^n E_j + \alpha \sum_{j=1}^n T_j + \alpha \sum_{j=1}^n d_j$. Since the job earlinesses and tardinesses are nonnegative by definition, we have that $f(\sigma) \geq \alpha \sum_{j=1}^n d_j$ for each $\sigma \in \Omega$.

We gain more insight if we derive this bound in the following way. Suppose the machine can process any number of jobs at the same time; this is a relaxation of the limited capacity of the machine. As $\alpha < \beta$, the optimal schedule has $C_j = d_j$ for each J_j; this gives rise to the lower bound $\alpha \sum_{j=1}^n d_j$. If no jobs overlap in their execution, then this schedule is feasible and hence optimal for the original problem.

We can improve on the lower bound $\alpha \sum_{j=1}^n d_j$ by taking the overlap between jobs into consideration. Overlap of J_j and J_k ($J_j \neq J_k$) occurs if the intervals $[d_j - p_j, d_j]$ and $[d_k - p_k, d_k]$ overlap. Simple computations show that $c_{jk} = \gamma \max\{0, d_j - (d_k - p_k)\}$ is the *minimal additional* cost to execute J_j immediately before J_k, where $\gamma = \min\{\alpha, \beta - \alpha\}$. Let $\sigma(i) = j$ denote that J_j occupies the ith position in the sequence σ. For any schedule σ, we have that $f(\sigma) \geq \alpha \sum_{j=1}^n d_j + \sum_{j=1}^{n-1} c_{\sigma(j)\sigma(j+1)}$; the last term is the length of the Hamiltonian path $\sigma(1) \cdots \sigma(n)$. The following procedure shows that in our application the Hamiltonian path problem of finding a sequence σ with minimum $\sum_{j=1}^{n-1} c_{\sigma(j)\sigma(j+1)}$ is solvable in $O(n \log n)$ time.

Partition the set of jobs into a set of clusters $Q_1, \ldots, Q_m$ such that each overlapping pair of jobs belongs to the same cluster. Since each conflict between two jobs that are not in the same cluster is settled at zero cost, we have that the length of the optimal Hamiltonian path is equal to the sum of the lengths of the optimal Hamiltonian paths within each cluster. Consider any cluster Q_i; let H_i denote the optimal Hamiltonian path within Q_i and let $c(H_i)$ denote its cost. It is readily checked that $c(H_i) = \gamma(p(Q_i) - \max_{J_j, J_k \in Q_i, J_k \neq J_j} c_{jk})$, from which we obtain the lower bound $\alpha \sum_{j=1}^n d_j + \sum_{i=1}^m c(H_i)$.

3.3 Lower Bound 3: Relax the Due Dates

Suppose the due dates have been replaced by a due date d common to all jobs. Consider the following *common due date problem,* referred to as problem (CD): for a given d, determine a schedule that minimizes

$$(\beta - \alpha) \sum_{j=1}^n E_j + \alpha \sum_{j=1}^n T_j + \alpha n d - \beta \sum_{j=1}^n \max\{0, d - d_j\}.$$

For any d, the optimal solution value is a lower bound for the original problem, since

$$f(\sigma) = \alpha \sum_{j=1}^n C_j + \beta \sum_{j=1}^n \max\{0, d_j - C_j\}$$

$$\geq \alpha \sum_{j=1}^n C_j + \beta \sum_{j=1}^n \max\{0, d - C_j\}$$

$$- \beta \sum_{j=1}^n \max\{0, d - d_j\}$$

$$= (\beta - \alpha) \sum_{j=1}^n E_j + \alpha \sum_{j=1}^n T_j$$

$$+ \alpha n d - \beta \sum_{j=1}^n \max\{0, d - d_j\}.$$

There are two issues involved: (i) how to solve problem (CD)?, and (ii) how to find the value d maximizing the lower bound?

Problem (CD) consists of two parts. The first part is the problem of minimizing $(\beta - \alpha) \sum_{j=1}^n E_j + \alpha \sum_{j=1}^n T_j$. If the machine is only available from time 0 onward and if d is given, then this problem is NP-hard (Hall, Kubiak, and Sethi[7]; Hoogeveen and van de Velde[9]). However, a strong lower bound $L(d)$ is derived by applying Lagrangian relaxation (see Hoogeveen, Oosterhout, and van de Velde[8]). The second part is the problem of maximizing the function $G: d \rightarrow \alpha n d - \beta \sum_{j=1}^n \max\{0, d - d_j\}$; this problem is solvable in polynomial time. Rather than solving problem (CD) to optimality and finding the best d, we maximize the lower bound $L(d) + G(d)$ over d.

The function $L: d \rightarrow L(d)$ is continuous and piecewise linear, with at most $\min\{n^2, n\alpha\}$ breakpoints. The function $G: d \rightarrow G(d)$ is also continuous and piecewise linear; the breakpoints correspond to the values $d = d_j$, for $j = 1, \ldots, n$. The lower bound $L(d) + G(d)$ is therefore also continuous and piecewise linear in d; the value d maximizing this lower bound is found at a breakpoint. Hence, maximizing $L(d) + G(d)$ over d is achieved in $O(n^2)$ time.

There are two ways to implement this lower bound procedure in a node of the search tree, depending on whether we take the possible overlap between π, the partial schedule associated with the node, and the optimal schedule for the common due date problem into account. If we do not, then we get the lower bound $f(\pi) + c(\mathscr{J} - \pi)$, where $c(\mathscr{J} - \pi)$ denotes the optimal solution value for the common due date problem for the jobs in $\mathscr{J} - \pi$. If we take the overlap into account, then we need to specify which part of π is affected by the optimal schedule for the common due date problem; let this be π_1, that is, $\pi = \pi_1\pi_2$. Given the part π_1 that we want to include, we compute this restricted common due date problem in the following way. First of all, we require that d is common to each $J_j \notin \pi_2$. Subsequently, we solve the common due date problem under the condition that the jobs in π_1 retain their positions in the sequence. This problem is solved using the concept of positional weights; see Emmons[2] and Hoogeveen and van de Velde.[10]

The common due date lower bound can only be used in conjunction with Theorem 2 if the lower bound is independent from the partial sequence $j\pi$. It is effective if the due dates are close to each other.

In the same spirit, we can consider the special case of $1\|\alpha \sum_{j=1}^{n} C_j + \beta \sum_{j=1}^{n} E_j$, where all jobs have equal slack time s; i.e., $d_j - p_j = s$ for each J_j $(j = 1, \ldots, n)$. This problem has the same features as the common due date problem. The best Lagrangian lower bound is also computed in $O(n \min\{\alpha, n\})$ time, and there are the same options to implement the lower bound.

3.4 Lower Bound 4: Relax the Processing Times

Define $p_{\min} = \min_{1 \le j \le n} p_j$. The optimal solution value of the relaxed problem $1|p_j = p_{\min}|\alpha \sum_{j=1}^{n} C_j + \beta \sum_{j=1}^{n} E_j$ provides a lower bound for the original problem: each set of job completion times that is feasible for the original problem is also feasible for the relaxed problem and has equal cost. Theorem 3 indicates that this relaxed problem is solved by scheduling the jobs in *earliest-due-date* order (i.e., the sequence with the jobs in order of nondecreasing due dates); see also Garey, Tarjan, and Wilfong.[5]

Given a partial schedule π, let σ be the earliest-due-date sequence for the jobs in $\mathcal{J} - \pi$, and let $h(\sigma)$ be the optimal solution value for the relaxed problem. Disregarding π, we get the lower bound $f(\pi) + g(\sigma)$. We can marginally improve on this lower bound. Suppose we have reindexed the jobs in order of nondecreasing due dates. Corollary 4 indicates that J_n is also scheduled last if we put its processing time equal to $\min\{p_n, p_{\min} + d_n - d_{n-1}\}$. An improved lower bound is therefore given by $f(\pi) + h(\sigma) + \alpha[\min\{p_n, p_{\min} + d_n - d_{n-1}\} - p_{\min}]$.

If the execution of jobs in σ overlaps with the execution of jobs in π, then it pays to take π into account. The lower bound is then equal to the cost for the sequence of $\sigma\pi$ with the jobs in π still having their original processing times.

Both bounds are computed in $O(n^2)$ time, the time needed to compute the optimal schedule for a given sequence, and dominate the lower bound $\alpha \sum_{j=1}^{n} d_j$. Only the first version can be used in conjunction with Theorem 2. The bounds are only effective if the processing times are close to each other.

3.5 Lower Bound 5: Lagrangian Relaxation

Our last lower bound approach is based on Lagrangian relaxation. For the proofs of the theorems in this section, we refer to Hoogeveen and van de Velde.[10] The problem of minimizing total cost, referred to as problem (P), can be formulated as follows. Determine values C_j and E_j $(j = 1, \ldots, n)$ that minimize

$$\alpha \sum_{j=1}^{n} C_j + \beta \sum_{j=1}^{n} E_j$$

subject to

$$E_j \ge 0, \quad \text{for} \quad j = 1, \ldots, n, \tag{3}$$

$$E_j \ge d_j - C_j, \quad \text{for} \quad j = 1, \ldots, n, \tag{4}$$

$$C_j \ge C_k + p_j \quad \text{or} \quad C_k \ge C_j + p_k, \tag{5}$$

$$\text{for} \quad j, k = 1, \ldots, n, j \ne k,$$

$$C_j \ge p_j, \quad \text{for} \quad j = 1, \ldots, n. \tag{6}$$

The conditions (3) and (4) reflect the definition of job earliness, while the conditions (5) ensure that the machine executes at most one job at a time. The conditions (6) express that the machine is available only from time 0 onwards.

We introduce a nonnegative vector $\lambda = (\lambda_1, \ldots, \lambda_n)$ of Lagrangian multipliers in order to dualize the conditions (3). For any given vector $\lambda \ge 0$, the Lagrangian problem, referred to as problem (L_λ) is to determine the value $L(\lambda)$, which is the minimum of

$$\alpha \sum_{j=1}^{n} C_j + \sum_{j=1}^{n} (\beta - \lambda_j) E_j$$

subject to the conditions (4), (5), and (6). We know that for any given $\lambda \ge 0$ the value $L(\lambda)$ provides a lower bound to problem (P). If $\beta - \lambda_j < 0$ for some J_j, we get $E_j = \infty$, which disqualifies the lower bound. We therefore assume that

$$\lambda_j \le \beta, \quad \text{for} \quad j = 1, \ldots, n. \tag{7}$$

This, in turn, implies that there exist an optimal solution to the Lagrangian problem in which conditions (4) hold with equality: $E_j = d_j - C_j$ for each j $(j = 1, \ldots, n)$. Hence, the Langrangian problem, referred to as problem (L_λ), transforms into the problem of minimizing

$$\sum_{j=1}^{n} (\alpha - \beta + \lambda_j) C_j + \sum_{j=1}^{n} (\beta - \lambda_j) d_j$$

subject to the conditions (5) and (6). If $\alpha - \beta + \lambda_j < 0$ for some J_j, we get $C_j = \infty$, which makes the lower bound rather weak. As demonstrated at the beginning of Section 3, however, we can determine an upper bound T on the job completion times:

$$C_j \le T, \quad \text{for} \quad j = 1, \ldots, n. \tag{8}$$

Although the conditions (8) are redundant for the primal problem (P), they are essential to admit values $\lambda_j < \beta - \alpha$. We determine the sets of jobs $\mathcal{J}^+ = \{J_j|\lambda_j > \beta - \alpha\}$, $\mathcal{J}^- = \{J_j|\lambda_j < \beta - \alpha\}$, and $\mathcal{J}^0 = \{J_j|\lambda_j = \beta - \alpha\}$. The following theorem stipulates that problem (L_λ) with conditions (8) is solved by a simple extension of Smith's rule (Smith[14]) for solving the $1\|\Sigma w_j C_j$ problem; the proof proceeds by an elementary interchange argument.

Theorem 8: *Problem (L_λ) with the additional conditions (8) is solved by scheduling the jobs in $\mathcal{J}^+$ in nonincreasing order of ratios $(\alpha - \beta + \lambda_j)/p_j$ in the interval $[0, p(\mathcal{J}^+)]$ and the jobs in $\mathcal{J}^-$ in nonincreasing order of ratios $(\alpha - \beta + \lambda_j)/p_j$ in the interval*

$[T - p(\mathcal{J}^-), T]$. *The jobs in* $\mathcal{J}^0$ *can be scheduled in any order in the interval* $[p(\mathcal{J}^+), T - p(\mathcal{J}^-)]$.

We are interested in determining the vector $\lambda^* = (\lambda_1^*, \ldots, \lambda_n^*)$ of Lagrangian multipliers that gives the best Lagrangian lower bound. The vector λ^* stems from solving the *Lagrangian dual problem,* referred to as problem (D):

$$\max\{L(\lambda) | 0 \leqslant \lambda_j \leqslant \beta, \quad \text{for} \quad j = 1, \ldots, n\}.$$

Problem (D) is solvable to optimality in polynomial time by use of the ellipsoid method; see van de Velde.[15] Since the ellipsoid method is very slow in practice, we take our resort to an approximation algorithm for problem (D).

First, we identify the *primitive directional derivatives*. In the solution to the Lagrangian problem (L_λ), the position of J_j depends on the ratio $(\alpha - \beta + \lambda_j)/p_j$; we call this ratio the *relative weight* of J_j. The larger this relative weight, the smaller the completion time of J_j. If other jobs have precisely the same relative weight as J_j, then the exact position of J_j is determined by settling ties. Let now $C_j^+(\lambda)$ denote the earliest possible completion time of J_j in an optimal schedule for problem (L_λ); let $C_j^-(\lambda)$ denote the latest possible completion time of J_j in an optimal schedule for problem (L_λ). If we increase λ_j by $\epsilon > 0$, then we can choose ϵ small enough to make sure that at least one optimal schedule for problem (L_λ) remains optimal; for a proof, see van de Velde.[15] In fact, all such optimal schedules must have J_j completed on time $C_j^+(\lambda)$. If we increase λ_j by such a sufficiently small $\epsilon > 0$, then the Lagrangian objective value is affected by $\epsilon(C_j^+(\lambda) - d_j)$. The primitive directional derivative for increasing λ_j, as denoted by $l_j^+(\lambda)$, is therefore simply

$$l_j^+(\lambda) = C_j^+(\lambda) - d_j, \quad \text{for} \quad j = 1, \ldots, n.$$

Hence, if $l_j^+(\lambda) > 0$, then increasing λ_j is an ascent direction: we get an improved lower bound by moving some scalar step size along this direction. In a similar fashion, we derive that the primitive directional derivative for decreasing λ_j, denoted by $l_j^-(\lambda)$, is

$$l_j^-(\lambda) = d_j - C_j^-(\lambda), \quad \text{for} \quad j = 1, \ldots, n.$$

If $l_j^-(\lambda) > 0$, then decreasing λ_j is an ascent direction. Note that directional derivatives may not exist at the boundaries of the feasible region of λ; for instance, $l_j^-(\lambda)$ is undefined for $\lambda = (0, \ldots, 0)$, for any $j = 1, \ldots, n$.

Second, we determine an appropriate step size $\Delta > 0$ to move by along a chosen ascent direction. We compute the step size that takes us to the first point where the corresponding primitive directional derivative is no longer positive. If no such point exists, then we choose the step size as large as possible while maintaining feasibility.

Suppose $l_j^+(\lambda) > 0$: J_j is tardy in any optimal schedule for problem (L_λ). Increasing λ_j, thereby putting J_j earlier in the schedule, is an ascent direction. We distinguish the cases $p_j - d_j > 0$, $p_j - d_j = 0$, and $p_j - d_j < 0$. Consider the case $p_j - d_j > 0$. Hence, J_j is unavoidably tardy, and $l_j^+(\lambda) > 0$ for all $\lambda \geqslant 0$ with $\lambda_j < \beta$. Therefore, we take the step size $\Delta = \beta - \lambda_j$. Accordingly, we must also have that $\lambda_j^* = \beta$; otherwise, increasing λ_j^* would be an ascent direction. If $p_j = d_j$, then there exists an optimal solution to problem (D) with $\lambda_j^* = \beta$. Find $\mathcal{T} = \{J_j | p_j \leqslant d_j\}$. We have proven the following result.

Theorem 9: *There is an optimal solution for problem* (D) *with* $\lambda_j^* = \beta$ *for each* $J_j \in \mathcal{T}$.

Suppose now $p_j < d_j$. The step size Δ must satisfy $\lambda_j + \Delta \leqslant \beta$. We identify the first job in the schedule, say, J_k, for which $C_k - p_k + p_j \leqslant d_j$. Since $p_j < d_j$, such an J_k always exists. If J_j is scheduled in J_k's position, then J_j is not tardy. Hence, if there were no upper bound on λ, then increasing λ_j would be an ascent direction up to the point where the relative weight of J_j becomes equal to the relative weight of J_k. Hence, the maximum step size along this ascent direction is the largest value Δ such that

$$\frac{\alpha - \beta + \lambda_j + \Delta}{p_j} \leqslant \frac{\alpha - \beta + \lambda_k}{p_k}, \quad \text{and} \quad \lambda_j + \Delta \leqslant \beta.$$

Let now $\bar{\lambda} = (\lambda_1, \ldots, \lambda_j + \Delta, \ldots, \lambda_n)$. Suppose, $\bar{\lambda}_j + \Delta < \beta_j$. Since the relative weights for all jobs but J_j have remained the same, optimal solutions for the problems ($L_{\bar{\lambda}}$) and (L_λ) exist with the same jobs scheduled before J_k. Now J_j and J_k have equal relative weights: in any optimal solution to problem ($L_{\bar{\lambda}}$), J_j can be scheduled before J_k or after J_k. If J_j is scheduled before J_k, then J_j is not tardy; if J_j is scheduled after J_k, then J_j is not early. Hence, we have that $C_j^+(\bar{\lambda}) \leqslant d_j \leqslant C_j^-(\bar{\lambda})$; the step size Δ has taken us to the first point where the primitive directional derivative for increasing λ_j is no longer positive. If $\bar{\lambda}_j = \beta$, then the step size has been chosen as large as possible.

Suppose now $l_j^-(\lambda) < 0$: J_j is early in any optimal schedule for problem (L_λ). Decreasing λ_j, thereby deferring J_j, is an ascent direction. We distinguish the cases $d_j > T$, $d_j = T$, and $d_j < T$. Consider the case $d_j > T$; hence, J_j is unavoidably early, and $l_j^-(\lambda) > 0$ for all λ with $\lambda_j > 0$. Therefore, we choose the step size as large as possible: $\Delta = \lambda_j$. Accordingly, we also must have that $\lambda_j^* = 0$; otherwise, decreasing λ_j^* would be an ascent direction. If $d_j = T$, then there exists an optimal schedule to problem (D) with $\lambda_j^* = 0$. Identify $\mathscr{E} = \{J_j | d_j \geqslant T\}$. We have proven the following result.

Theorem 10: *There is an optimal solution for problem* (D) *with* $\lambda_j^* = 0$ *for each* $J_j \in \mathscr{E}$.

Consider now the case $d_j < T$. The procedure to compute the appropriate step size Δ proceeds in a similar fashion as above. We identify some J_k as the first job in the schedule with $C_k \geqslant d_j$. If J_j is scheduled in J_k's position, then J_j is not early. Hence, if there were no lower bound on λ, then decreasing λ_j would be an ascent direction up to the point where the relative weight of J_j becomes equal to the relative weight of J_k. This implies that the maximum step size along this ascent direction is the largest value Δ for which

$$\frac{\alpha - \beta + \lambda_j - \Delta}{p_j} \geqslant \frac{\alpha - \beta + \lambda_k}{p_k}, \quad \text{and} \quad \lambda_j - \Delta \geqslant 0.$$

Let $\bar{\lambda} = (\lambda_1, \ldots, \lambda_j - \Delta, \ldots, \lambda_n)$. Suppose $\bar{\lambda}_j > 0$. Since the relative weights for all jobs but J_j have remained the same, optimal solutions for the problems ($L_{\bar{\lambda}}$) and (L_λ) exist with the same jobs scheduled after J_k. Since J_j and J_k have now equal weights, J_j can be scheduled after J_k or before J_k in any optimal schedule for problem $(\mathrm{L}_{\bar\lambda})$. If J_j is scheduled after J_k, then J_j is not early; if J_j is scheduled before J_k, then J_j is not tardy. Hence, we find that $C_j^+(\bar\lambda) \le d_j \le C_j^-(\bar\lambda)$. If $\bar\lambda_j = 0$, then the step was taken as large as possible.

Termination of the ascent direction procedure occurs at some $\bar\lambda$ where all existing primitive directional derivatives are nonpositive. If all primitive directional derivatives exist at such a $\bar\lambda$, we have

$$C_j^+(\bar\lambda_j) \le C_j^-(\bar\lambda) \quad \text{for} \quad j = 1, \ldots, n.$$

These termination conditions also apply to λ^*, since they are necessary for optimality. They are, however, not sufficient for optimality; hence, termination may occur having $\bar\lambda \ne \lambda^*$, i.e., before finding the optimal vector of Lagrangian multipliers. Before implementing the ascent direction algorithm, we make use of this fact to decompose the Lagrangian dual problem (D) into two subproblems. This decomposition is achieved by partitioning $\mathcal{J}$ into four subsets, including the sets $\mathcal{T}$ and $\mathcal{E}$ we already identified.

Consider some job $J_j \in \mathcal{J} - \mathcal{E}$ with $d_j > p(\mathcal{J} - \mathcal{E})$. If $\lambda_j > \beta - \alpha$, then J_j will be early in any optimal solution to problem (L_λ). This means that $l_j^-(\lambda) > 0$, and hence we must have that $0 \le \lambda_j^* \le \beta - \alpha$. The set $\mathcal{F}$ of jobs that share this property is determined by the following procedure.

Partitioning Algorithm 1

Step 1: $\mathcal{F} \leftarrow \varnothing$, $k \leftarrow 1$; reindex the jobs in $\mathcal{J} - \mathcal{E}$ in order of nonincreasing due dates.
Step 2: If $k > n - |\mathcal{E}|$ or if $d_k < p(\mathcal{J} - \mathcal{E} - \mathcal{F})$, then stop. Else $\mathcal{F} \leftarrow \mathcal{F} \cup \{J_k\}$.
Step 3: Put $k \leftarrow k + 1$; go to Step 2.

Suppose some job $J_j \in \mathcal{F}$ exists with $d_j > T - p(\mathcal{E})$. If we let $\lambda_j = \beta - \alpha$, then $C_j^-(\lambda) < d_j$; hence, decreasing λ_j is an ascent direction. Decreasing λ_j gives $(\alpha - \beta + \lambda_j)/p_j < 0$, as a result of which the execution of J_j interferes with the execution of the jobs in $\mathcal{E}$. We now partition the set $\mathcal{F}$ into subsets $\mathcal{F}_1$ and $\mathcal{F}_2$ ($\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$) such that $d_j \le T - p(\mathcal{E} \cup \mathcal{F}_2)$ for each $J_j \in \mathcal{F}_1$, and such that $d_j > T - p(\mathcal{E} \cup \mathcal{F}_2)$ for each $J_j \in \mathcal{F}_2$. To achieve this, we use the following partitioning procedure; it is similar to the first one.

Partitioning Algorithm 2

Step 1: Put $\mathcal{F}_2 \leftarrow \varnothing$, let $P \leftarrow T - p(\mathcal{E})$, and reindex the jobs in $\mathcal{F}$ according to nonincreasing due dates. Let $k \leftarrow 1$.
Step 2: If $k > |\mathcal{F}|$, then stop. If $d_k \le P$, then let $\mathcal{F}_1 \leftarrow \{J_k, \ldots, J_{|\mathcal{F}|}\}$, and stop. Otherwise, $\mathcal{F}_2 \leftarrow \mathcal{F}_2 \cup \{J_k\}$, and set $P \leftarrow P - p_k$.
Step 3: Set $k \leftarrow k + 1$; go to Step 2.

Theorem 11: *For each $J_j \in \mathcal{F}_1$, we have that $\lambda_j^* = \beta - \alpha$.*

At this stage, we can decompose the Lagrangian dual problem (D) into two subproblems. Let $\mathcal{R} = \mathcal{J} - \mathcal{T} - \mathcal{E} - \mathcal{F}$. Since $(\alpha - \beta + \lambda_j^*)/p_j = 0$ for each $J_j \in \mathcal{F}_1$, the jobs in $\mathcal{F}_1$ do not interfere with the execution of the other jobs. We have, however, that both $\mathcal{T}$ and $\mathcal{R}$ and $\mathcal{E}$ and $\mathcal{F}_2$ interfere with each other. Hence, we have to solve the dual problem restricted to the sets $\mathcal{T}$ and $\mathcal{R}$ and the dual problem restricted to the sets $\mathcal{F}_2$ and $\mathcal{E}$. In each optimal schedule for problem (D), the jobs in $\mathcal{T}$ and $\mathcal{R}$ are scheduled in the interval $[0, p(\mathcal{T} \cup \mathcal{R})]$, and the jobs in $\mathcal{F}$ and $\mathcal{E}$ are scheduled in the interval $[T - p(\mathcal{E} \cup \mathcal{F}_2), T]$. We give step-wise descriptions of the ascent direction algorithms for these two subproblems. Both are based upon the primitive directional derivatives and the step sizes we discussed earlier. The jobs in $\mathcal{F}_1$ are scheduled somewhere in the interval $[p(\mathcal{T} \cup \mathcal{R}), T - p(\mathcal{E} \cup \mathcal{F}_2)]$; they are left out of consideration. We introduce some new notation. Let $(\mathrm{L}_\lambda^{\mathcal{R}\cup\mathcal{T}})$ and $(\mathrm{L}_\lambda^{\mathcal{E}\cup\mathcal{F}_2})$ denote the Lagrangian problem restricted to the set $\mathcal{R} \cup \mathcal{T}$ and to the set $\mathcal{E} \cup \mathcal{F}_2$; let $\mathrm{L}^{\mathcal{R}\cup\mathcal{T}}(\lambda)$ and $\mathrm{L}^{\mathcal{E}\cup\mathcal{F}_2}(\lambda)$ denote their optimal solution values.

Ascent Direction Algorithm for the Set $\mathcal{R} \cup \mathcal{T}$

Step 1: For each $J_j \in \mathcal{T}$, set $\lambda_j \leftarrow \lambda_j^* = \beta$; for each $J_j \in \mathcal{R}$, put $\lambda_j \leftarrow \beta$. Solve $(\mathrm{L}_\lambda^{\mathcal{R}\cup\mathcal{T}})$, settling ties arbitrarily; compute the job completion times.
Step 2: For each $J_j \in \mathcal{R}$, do the following:

(a) If $C_j^-(\lambda) < d_j$, identify J_k as the first job in the schedule with $C_k \ge d_j$. Compute the largest value Δ such that

$$\frac{\alpha - \beta + \lambda_j - \Delta}{p_j} \ge \frac{\alpha - \beta + \lambda_k}{p_k}, \quad \text{and} \quad \lambda_j - \Delta \ge \beta - \alpha.$$

Decrease λ_j by Δ, reposition J_j according to its new relative weight, and update the job completion times.

(b) If $C_j^+(\lambda) > d_j$, identify J_k as the first job in the schedule with $C_k - p_k + p_j \le d_j$. Compute the largest value for Δ such that

$$\frac{\alpha - \beta + \lambda_j + \Delta}{p_j} = \frac{\alpha - \beta + \lambda_k}{p_k}, \quad \text{and} \quad \lambda_j + \Delta \le \beta.$$

Increase λ_j by Δ, reposition J_j according to its new relative weight, and update the job completion times.

Step 3: If no multiplier adjustment has taken place, then compute $\mathrm{L}^{\mathcal{R}\cup\mathcal{T}}(\lambda)$ and stop. Otherwise, go to Step 2.

Theorem 12: *The procedure described above generates a series of monotonically increasing values $L^{\mathcal{R}\cup\mathcal{T}}(\lambda)$.*

Ascent Direction Algorithm for the Set $\mathcal{E} \cup \mathcal{F}_2$

Step 1: Set $\lambda_j \leftarrow \beta - \alpha$ for each $J_j \in \mathcal{F}_2$, and $\lambda_j \leftarrow \lambda_j^* = 0$ for each $J_j \in \mathcal{E}$. Solve $(\mathrm{L}_\lambda^{\mathcal{E}\cup\mathcal{F}_2})$, settling ties arbitrarily; compute the job completion times.
Step 2: For each $J_j \in \mathcal{F}_2$, do the following:

(a) If $C_j^-(\lambda) < d_j$, identify J_k as the first job in the schedule with $C_k \le d_j$. Compute the largest value Δ such that

$$\frac{\alpha - \beta + \lambda_j - \Delta}{p_j} \ge \frac{\alpha - \beta + \lambda_k}{p_k}, \quad \text{and} \quad \Delta \le \lambda_j.$$

Decrease λ_j *by* Δ, reposition J_j according to its new relative weight, and update the job completion times.

(b) If $C_j^+(\lambda) > d_j$, identify J_k as the first job in the schedule with $C_k \le d_j + p_k - p_j$. Compute the largest value for Δ such that

$$\frac{\alpha - \beta + \lambda_j + \Delta}{p_j} = \frac{\alpha - \beta + \lambda_k}{p_k}, \quad \text{and} \quad \lambda_j + \Delta \le \beta - \alpha.$$

Increase λ_j by Δ, reposition J_j according to its new relative weight, and update the job completion times.

Table I. Properties of the Proposed Lower Bounds

type of relaxation	complexity	effective when	partitioning strategy
objective function	$O(n)$	little idle time	surely early jobs
machine capacity	$O(n \log n)$	conflicting jobs	clustering
due dates	$O(n^2)$	d_j almost equal	clustering
processing times	$O(n^2)$	p_j almost equal	clustering
Lagrangian	$O(n \log n + In)$	little idle time	surely early jobs

Step 3: If no multiplier adjustment has taken place, then compute $L^{\mathscr{E} \cup \mathscr{F}_2}(\lambda)$ and stop. Otherwise, go to Step 2.

Theorem 13: *The procedure described above generates a series of monotonically increasing values* $L^{\mathscr{E} \cup \mathscr{F}_2}(\lambda)$.

For each $J_j \in \mathscr{J} - \mathscr{F}_1$, let C_j and $\bar{\lambda}_j$ denote the completion time and the Lagrangian multiplier upon termination of the appropriate ascent direction algorithm. We note that $\bar{\lambda}_j = \beta_j$ for each $J_j \in \mathscr{T}$, $\bar{\lambda}_j = \beta - \alpha$ for each $J_j \in \mathscr{F}_1$, and $\bar{\lambda}_j = 0$ for each $J_j \in \mathscr{E}$. Hence, the overall Lagrangian lower bound is given by

$$L(\bar{\lambda}) = \sum_{J_j \in \mathscr{T}} \alpha C_j + \sum_{J_j \in \mathscr{F}_1} \alpha d_j + \sum_{J_j \in \mathscr{E}} [(\alpha - \beta)C_j + \beta d_j]$$
$$+ \sum_{J_j \in \mathscr{R} \cup \mathscr{F}_2} [(\alpha - \beta + \bar{\lambda}_j)C_j - (\beta - \bar{\lambda}_j)d_j].$$

The Lagrangian lower bound performs well if little idle time is involved. Either ascent direction algorithm is an iterative non-polynomial method, taking $O(n \log n + In)$ time with I the number of iterations.

3.6 Overview of the Bounds

In this subsection, we tabulate the lower bounds presented, the time needed to compute them, the conditions under which they are expected to be effective, and the partitioning strategy they can be used in tandem with. Roughly speaking, we may say that we have two types of bounds available: those that perform well, if there is little idle time involved, or if there are many surely early jobs, and those that perform well, if all jobs are conflicting, or if the clusters hardly overlap. Beforehand, it is impossible to tell whether the aggregate bound outperforms the best individual bound. Also, it is possible to be more conclusive as to which lower bound will be strongest for a given cluster. First, it is hard to tell in advance how much and when idle time will be inserted. Second, 'conflicting jobs' is a broad notion that is only well-defined at the extremes: if all the ideal execution intervals $[d_j - p_j, d_j]$ overlap, then we have a relatively easy problem; if there is no overlap, then the problem even becomes trivial.

On the basis of Table 1, we may anticipate that our best lower bound will be weak for instances with overlapping clusters in which there is a great deal of idle time involved.

4. Computational Results

The algorithm was coded in the computer language C; the experiments were conducted on a Compaq-386/20 Personal Computer. The algorithm was tested on instances with 8, 10, 12, 15, and 20 jobs. The processing times were generated from the uniform distribution [10, 100]. The due dates were generated from the uniform distribution $[P(1 - T - R/2), P(1 - T + R/2)]$, where $P = \Sigma_{j=1}^{n} p_j$ and where R and T are parameters. For both parameters, we considered the values 0.2, 0.4, 0.6, 0.8, and 1.0. This procedure to generate due dates parallels the procedure described by Potts and van Wassenhove[13] for the weighted tardiness problem. For each combination of T, P, and n, we generated 5 instances. Each instance was considered with $\alpha = 1$ and with β running from 2 to 5.

The general impression was that instances become more difficult with smaller values of T and R. In view of the discussion in Section 3.6, this could be expected. A small value of T induces relatively large due dates, implying that there is quite some idle time involved. A small value of R induces due dates that are close to each other; it is then harder to partition the jobs. We also found that the problems get easier with larger β. This may seem anomalous: a large value of β implies that earliness is severely penalized, that many jobs therefore will be tardy, and that hence a lot of idle time will be inserted. In this case, however, the dominance rules in Section 3 become more effective. Summarizing, we have that the instances with $T = 0.2$, $R = 0.2$, and small β are the hardest, and that the instances with $T = 1.0$, $R = 1.0$, and $\beta = 5$ are the easiest.

Table 2 exhibits a summary of our computational results; we only report the results for the instances with T and R equal. For each combination of n ($n = 8, 10, 12, 15, 20$), of T and R ($T = R = 0.2, 0.4, 0.6, 0.8, 1.0$), and of β ($\beta = 2, 3, 4, 5$), we present the average number of nodes and the average number of seconds; the average was computed over 5 instances. All averages were rounded up to the nearest integer. The sign '—' indicates that not all instances of this particular combination could be solved without examining more than 100,000 nodes.

We conclude that instances with up to 10 jobs are easy. For $n = 12$, the instances with $T = R = 0.2$ require already considerable effort. For $n = 20$, only the choice $T = R = 1.0$ induces instances that are solvable within reasonable time limits. It is likely, however, that the performance of the algorithm can be considerably enhanced by fine-tuning the

Table II. Computational Results

		$\beta = 2$		$\beta = 3$		$\beta = 4$		$\beta = 5$	
n	T, R	nodes	sec	nodes	sec	nodes	sec	nodes	sec
8	0.2	417	2	406	2	301	2	58	1
8	0.4	131	1	198	1	185	1	31	1
8	0.6	34	1	48	1	29	1	5	1
8	0.8	23	1	37	1	14	1	8	1
8	1.0	20	1	36	1	33	1	15	1
10	0.2	2,438	8	2,525	9	2,088	7	484	2
10	0.4	266	2	689	3	570	3	202	2
10	0.6	123	1	110	1	88	1	52	1
10	0.8	126	1	122	1	107	1	64	1
10	1.0	109	1	140	1	78	1	40	1
12	0.2	30,182	103	26,676	106	18,358	78	10,487	48
12	0.4	15,176	66	20,756	100	15,613	75	10,391	50
12	0.6	212	2	262	2	53	1	10	1
12	0.8	380	2	576	4	300	2	170	1
12	1.0	432	2	527	3	226	2	96	1
15	0.2	—	—	—	—	—	—	—	—
15	0.4	—	—	—	—	—	—	—	—
15	0.6	1,414	10	2,407	17	927	7	339	2
15	0.8	1,665	13	1,865	15	1,647	14	540	5
15	1.0	493	6	402	17	2,063	17	1,082	9
20	0.2	—	—	—	—	—	—	—	—
20	0.4	—	—	—	—	—	—	—	—
20	0.6	7,991	80	13,169	136	5,529	62	2,048	24
20	0.8	8,183	85	7,244	84	4,016	55	1,318	21
20	1.0	5,127	49	5,243	41	2,191	32	651	12

algorithm to these instances. In our implementation, all lower bounds are computed in each node of the tree, but, Lagrangian relaxation, for instance, is useless for instances with $T = R = 0.2$. For an arbitrary given instance, however, it is an interesting issue if we can decide beforehand which lower bound approach will not be effective.

We have made no attempt to compare the performance of our algorithm with the performance of the algorithms of Fry and Leong[3] and Fry, Leong, and Rakes.[4] As pointed out in the introduction, a standard integer linear programming code, in general, cannot compete with a specialized algorithm, which holds particularly true in scheduling. The branch-and-bound algorithm by Fry, Leong, and Rakes[4] uses two simple lower bounds: the one bound is based upon the scheduling of a single job only, which gives weak bounds; the other bound is based on minimizing $\Sigma\, C_j$, and gives a weaker version of our bound presented in Section 3.1.

5. Conclusions

Theoretically, the insertion of machine idle time is a valid instrument model to reduce earliness cost. We have considered an NP-hard machine scheduling problem in which the insertion of idle time may be advantageous, and we have presented a branch-and-bound algorithm for its solution. The allowance of machine idle time complicates the design of the algorithm substantially. Also, the performance of the algorithm is quite bleak: this is because it is very difficult to compute strong lower bounds.

Acknowledgment

The authors are grateful to Jan Karel Lenstra and the referees for their helpful comments. The research of the first author was supported by a grant from Netherlands Organization for Scientific Research (NWO).

References

1. K. BAKER and G. SCUDDER (1990). Sequencing with Earliness and Tardiness Penalties: A Review. *Operations Research 38*, 22–36.
2. H. EMMONS (1987). Scheduling to a Common Due Date on Parallel Uniform Processors. *Naval Research Logistics 34*, 803–810.
3. T.D. FRY and G.K. LEONG (1987). A Bi-criterion Approach to Minimizing Inventory Costs on a Single Machine when Early

Shipments are Forbidden. *Computers and Operations Research 14,* 363–368.

4. T.D. FRY, G.K. LEONG, and T.R. RAKES (1987). Single Machine Scheduling: A Comparison of Two Solution Procedures. *Omega 15,* 277–282.
5. M.R. GAREY, R.E. TARJAN, and G.T. WILFONG (1988). One-processor Scheduling with Symmetric Earliness and Tardiness Penalties. *Mathematics of Operations Research 13,* 330–348.
6. R.L. GRAHAM, E.L. LAWLER, J.K. LENSTRA, and A.H.G. RINNOOY KAN (1979). Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. *Annals of Discrete Mathematics 5,* 287–326.
7. N.G. HALL, W. KUBIAK, and S.P. SETHI (1991). Earliness–Tardiness Scheduling Problems II. Deviation of Completion Times about a Restrictive Common Due Date. *Operations Research 39,* 847–856.
8. J.A. HOOGEVEEN, H. OOSTERHOUT, and S.L. VAN DE VELDE (1994). New Lower and Upper Bounds for Scheduling around a Small Common Due Date. *Operations Research 42,* 102–110.
9. J.A. HOOGEVEEN and S.L. VAN DE VELDE (1991). Scheduling Around a Small Common Due Date. *European Journal of Operational Research 55,* 237–242.
10. J.A. HOOGEVEEN and S.L. VAN DE VELDE (1992). *Minimizing Total Inventory Cost on a Single Machine in Just-in-Time Manufacturing,* Memorandum COSOR 9220, Eindhoven University of Technology, Eindhoven, The Netherlands.
11. J.A. HOOGEVEEN and S.L. VAN DE VELDE (1996). Earliness–tardiness Scheduling Around Almost Equal Due Dates. *INFORMS Journal on Computing,* to appear.
12. J.J. KANET and D.P. CHRISTY (1984). Manufacturing Systems with Forbidden Early Order Departure. *International Journal of Production Research 22,* 41–50.
13. C.N. POTTS and L.N. VAN WASSENHOVE (1985). A Branch-and-Bound Algorithm for the Total Weighted Tardiness Problem. *Operations Research 33,* 363–377.
14. W.E. SMITH (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly 1,* 59–66.
15. S.L. VAN DE VELDE (1991). *Machine Scheduling and Lagrangian Relaxation,* Doctoral Thesis, CWI, Amsterdam.