MINIMIZING MEAN TARDINESS AND EARLINESS IN SINGLE MACHINE SCHEDULING PROBLEMS WITH UNEQUAL DUE-DATES

Yeong-Dae Kim
Department of Industrial Engineering
Korea Advanced Institute of Science and Technology
Yusong-gu, Daejon 305-701
Korea

Candace Arai Yano
Department of Industrial & Operations Engineering
University of Michigan
Ann Arbor, MI 48109-2117

Technical Report 92-44

July 1992

Minimizing Mean Tardiness and Earliness in Single Machine Scheduling Problems with Unequal Due-Dates

Yeong-Dae Kim
Department of Industrial Engineering
Korea Advanced Institute of Science and Technology
Yusong-gu, Daejon 305-701
Korea

Candace Arai Yano
Department of Industrial and Operations Engineering
The University of Michigan
Ann Arbor, Michigan 48109-2117
USA

July 1992

Abstract

We consider a single machine scheduling problem with the objective of minimizing the mean (or equivalently, total) tardiness and earliness when due dates may differ among jobs. Some properties of the optimal solution are discussed, and these properties are used to develop both optimal and heuristic algorithms. Results of computational tests indicate that optimal solutions can be found for problems with up to 20 jobs, and that two of the heuristic procedures provide optimal or very near optimal solutions in many instances.

1. Introduction

The widespread adoption of just-in-time (JIT) policies has led many manufacturing companies to develop scheduling policies with the goal of producing each customer order "at the necessary time." Translating this phrase into scheduling objectives leads us to consider both earliness and tardiness. (Throughout most of the literature, earliness and tardiness are defined as $\max(d_i - C_i, 0)$ and $\max(C_i - d_i, 0)$ respectively, where d_i is the due date and C_i is the completion time of job i.) There are, of course, real penalties (due to either out-of-pocket or opportunity costs) for both earliness and tardiness. For example, in production systems in which the shipping dates are specified, inventory carrying costs are incurred if jobs are finished earlier than their respective shipping dates, and shortage costs (explicit penalty costs, backorder costs, or transportation expediting costs, for example) are incurred if jobs are finished later than the shipping dates.

In this paper, our objective is to minimize mean tardiness and earliness. There are a number of reasons for considering such an objective. In some instances, it may be quite difficult to specify, for each job, the exact penalty as a function of its completion time. Our objective function provides a first-cut model for such situations. Our objective is also appropriate at an intermediate stage in a multi-stage manufacturing system, where the due dates are mutually agreed-upon completion times. In such situations, tardiness has the impact of delaying a downstream production stage, and earliness has the impact of forcing an upstream production stage to complete its processing earlier than originally anticipated. While it is clear that minimizing (or eliminating) both tardiness and earliness would be desirable, the choice of appropriate weights is not always obvious. It would be difficult, however, to argue for dramatically different weights when the key objective is one of coordination between production stages.

There are a number of research papers on scheduling problems involving tardiness and earliness costs. Sidney (1977) presents an optimal algorithm where the objective is to

minimize the maximum of the costs of earliness and those of tardiness, where the costs are nondecreasing functions of earliness and tardiness, respectively. A more efficient algorithm is developed for the same problem in Lakshminarayan *et al.* (1978).

Many papers consider the special case of a common due date, (i.e., due dates of all jobs are equal). Bagchi *et al.* (1987a, 1987b) consider the problem of minimizing mean squared deviation of completion times from a common due date. On the other hand, Kanet (1981), Sundararaghavan and Ahmed (1984), Bagchi *et al.* (1986, 1987a), Cheng (1987), Szwarc (1989), Ahmed and Sundararaghavan (1990), De *et al.* (1991), and Lee *et al.* (1991) use as the performance measure the absolute deviation of completion times from a common due date. In most of these papers, the weights for earliness and tardiness penalties are equal for all jobs. De *et al.* and Lee *et al.* allow the weights to differ, and Ahmed and Sundararaghavan consider the special case in which penalties are propotional to processing times of the jobs. Lee *et al.* also attempt to simultaneously minimize the number of tardy jobs. All of these papers assume the due date is given, with the exception of Cheng, where the due date is treated as a decision variable.

Hall and Posner (1991) prove that the common due-date problem is NP-complete when the weights differ among jobs. Hall *et al.* (1991) give an NP-completeness proof for the common due-date problem with equal weights when the due date is small enough so that not all jobs can be completed on time.

The more general problem (with unequal due dates) has been addressed in several papers. Ferris and Vlach (1992) discuss problems with the objectives of minimizing the maximum value of functions of earliness and tardiness and Gupta and Sen (1983) present algorithms for minimizing a quadratic function of job lateness. Seidmann *et al.* (1981), Fry *et al.* (1987a), Fry *et al.* (1987b), Abdul-Razaq and Potts (1988), Ow and Morton (1989), and Yano and Kim (1991) use objectives that are linear functions of earliness and tardiness.

Ow and Morton and Abdul-Razaq and Potts do not allow idle time between jobs. As pointed out in Baker and Scudder (1990), the assumption of no idle time is inconsistent with

the earliness/tardiness criterion. Since earliness is a nonregular measure of performance, delay schedules (schedules with idle time between jobs) may be optimal. Thus, we need to decide both the sequence and the exact timing of jobs. Yano and Kim, Fry *et al.* , and Garey *et al.* (1988) give algorithms for the optimal timing of a given sequence. A recent review of the literature on single machine earliness and tardiness problems appears in Baker and Scudder (1990).

In this paper, we address the single machine scheduling problem with the objective of minimizing mean tardiness plus earliness when due dates may differ among jobs. We do not impose the restriction of no idle times between jobs. This problem has been shown to be NP-complete by Garey *et al.* (1988). We develop an optimal implicit enumeration procedure and suggest several heuristics, since it is unlikely that optimal solutions can be found in a reasonable amount of time, except for small problems.

In the next section, we discuss methods for determining completion times of jobs when the sequence is given. In Section 3, we prove several properties of optimal job sequences. Using these properties, we develop a branch and bound algorithm and several heuristic algorithms. These are presented in sections 4 and 5. Section 6 contains results of computational tests on a wide range of randomly-generated problems. Finally, concluding remarks appear in Section 7.

2. Determining Optimal Completion Time of Each Job for a Given Sequence

When the sequence of jobs is given, an optimal schedule (timing) can be obtained easily by using a linear program. More efficient algorithms have been developed using special characteristics of this optimal timing problem (e.g., Yano and Kim 1986, Fry *et al.* 1987a,b, and Garey *et al.* 1986). Garey *et al.* give an algorithm that has complexity $O(n \log n)$, where n is the number of jobs to be scheduled. This efficiency is achieved through the use of efficient data structures.

For the optimal timing subroutine in our (branch and bound and heuristic) algorithms, we use a dynamic programming algorithm suggested in Yano and Kim (1986), which has $O(n^2)$ complexity in the worst case. (The procedure is also described in Yano and Kim 1991.) Computations in the DP are reduced considerably by taking advantage of properties of the objective function for a subsequence of jobs scheduled consecutively with no inserted idle time. In particular, the objective function for such a subsequence is a piecewise linear convex function of the starting time of the first job in the subsequence (or equivalently, completion time of the last job in the subsequence.) The computations can be further simplified by noting that as the starting time is delayed, the slope of the objective function changes only at points in time when one of the jobs changes from being early to being tardy. These properties are easiest to explain by way of an example.

Consider two jobs, say jobs 1 and 2, with processing times p_1 and p_2 and due dates d_1 and d_2, respectively. Suppose job 1 precedes job 2 and that $d_2 - d_1 < p_2$ (i.e., it is not possible to schedule the jobs without a penalty). While both jobs are early (i.e., $C_2 < d_2$), the objective function declines at a rate of 2 unit per unit time as the jobs are delayed. As soon as job 2 changes from being early to being tardy, and while job 1 is still early, the slope of the objective function is zero since job 2 is becoming tardier, but job 1 is becoming less early. As the jobs are delayed further, job 1 becomes tardy and the objective function starts to increase at a rate of 2 units per unit time.

This argument can be generalized to any number of jobs scheduled consecutively with no inserted idle time. If there are multiple "clusters" of jobs, then delaying the start of one cluster will initially incur costs reflected in its own piecewise linear convex function. When one cluster becomes concatenated to its successor cluster, the collective penalties for the two clusters continue to increase as a piecewise linear convex function as the starting time of the first job in the joint cluster is delayed.

These results imply that the objective function for any partial sequence is a piecewise linear convex function of the starting time of the first job in the partial sequence, given that

the (constrained) optimal timing decisions for all of the jobs in the partial sequence are used. We note that the optimal timing decision may not be unique. Because we construct the solution backward in our optimal solution procedure, we always choose the optimal solution with the latest starting time for the first job in the partial sequence. This provides the greatest flexibility to the jobs not yet scheduled, but with no additional cost.

While the algorithm of Yano and Kim (1991) has $O(n^2)$ complexity, it provides complete information about these piecewise linear convex functions, whereas the $o(n \log n)$ algorithm does not. This, in turn, allows us to derive better lower bounds. These bounds are described in Section 4.

3. Preliminaries

In this section, we first present some dominance rules for adjacent jobs, and then turn to the derivation of lower bounds for subsequences of jobs.

Proposition 1. *Consider two jobs, i and j, ($d_i \le d_j$). If there is a conflict between the two jobs when they are scheduled with $C_i=d_i$ and $C_j=d_j$, (that is, $d_j-p_j < d_i$), there exists an optimal schedule such that either $C_i=d_i$ or $C_j=d_j$.*

Proof. Obviously, there is no idle time between the two jobs in an optimal schedule. For either sequence, we only need to determine the completion of the first (or equivalently, the second) job. It is clear that any solution in which both jobs are early or both jobs are tardy is not optimal. It is easy to show that all of the remaining solutions (first job early, second job on time; first job early, second job tardy; first job on time, second job tardy) have the same objective value. Since this holds for both sequences, we can, without loss of generality, restrict consideration to solutions with $C_i=d_i$ or $C_j=d_j$. ■

Corollary 1. *If there is a conflict between two jobs, i and j, when they are scheduled with* $C_i=d_i$ *and* $C_j=d_j$, *the sum of tardiness and earliness is not less than the duration of the time conflict.*

Proof. Since an optimal schedule satisfies either $C_i=d_i$ or $C_j=d_j$, the result follows. ■

Using Proposition 1, we can obtain dominance conditions for two adjacent jobs as follows.

Proposition 2. *Consider two jobs, i and j,* $(d_i \le d_j)$. *If there is a conflict between the two jobs when they are scheduled with* $C_i = d_i$ *and* $C_j = d_j$: (1) *scheduling job i before job j is better if* $p_j - p_i < 2(d_j - d_i)$; and (2) *scheduling job j before job i is better if* $p_j - p_i > 2(d_j - d_i)$.

Proof. Let $TE(x,y)$ be the minimum value of the sum of tardiness and earliness of jobs x and y when x precedes y. From Proposition 1, we can obtain $TE(i,j)= p_j - (d_j - d_i)$ and $TE(j,i)= p_i + (d_j - d_i)$. Therefore, $TE(i,j) - TE(j,i) = p_j - p_i - 2(d_j - d_i)$. The result follows directly. ■

The above properties are useful in obtaining a lower bound on the sum of earliness and tardiness for a given set of jobs, which is given in the following theorem.

Theorem 1. *If there are conflicts among two or more jobs when a set of jobs is scheduled with* $C_i = d_i$ *for all i in the set, the sum of tardiness and earliness is greater than or equal to* $\sum_{k\ge2} (k-1)\, t_k$, *where* t_k *is the length of time during which k jobs overlap.*

Proof. Consider an arbitrary subset of these jobs such that each job conflicts with at least one other job in the subset, and none of the jobs conflicts with any job not in the subset when $C_i = d_i$ for all i. We can always partition the jobs into one or more subsets in this way. We show that $\sum_{i\in J^*} (T_i + E_i) \ge \sum_{k\ge2} (k-1)\, t_k$ for an arbitrary subset of jobs, J^*. By summing the bounds over all subsets, we can obtain a lower bound for the entire

schedule. Assume that the arbitrary subset has r jobs in it ($|J^*|=r$). Let (i) denote the index of the job in position i in the subsequence of these r jobs.

We first prove that for any sequence (1),(2),...,(r) of jobs in J^*, $\sum_{i\in J^*} (T_i + E_i) \geq p_{(2)} + p_{(3)} + \ldots + p_{(r)} - (d_{(r)} - d_{(1)})$. Treat jobs (2),(3),...,(r) as one job, A, with processing time $\sum_{i=2}^{r} p_{(i)}$ and due date $d_{(r)}$. Then, $T+E$ (the sum of tardiness and earliness) for jobs (1),(2),...,(r) is not less than $T+E$ for jobs (1) and A, since $T+E$ for jobs (2),...,(r–1) are nonnegative and $T+E$ for job (r) is equal to $T+E$ for job A. By Corollary 1, for two jobs (1) and A, a schedule where either $C_{(1)} = d_{(1)}$ or $C_A = d_{(r)}$ is optimal. In both cases, $T+E$ for jobs (1) and A is not less than $\sum_{i=2}^{r} p_{(i)} - (d_{(r)} - d_{(1)})$.

Next we prove that $p_{(2)} + \ldots + p_{(r)} - (d_{(r)} - d_{(1)}) \geq \sum_{k=2}^{r} (k-1)\, t_k$.

We have that

$$\begin{aligned} p_{(2)} + \ldots + p_{(r)} &= \sum_{k=1}^{r} p_k - p_{(1)} \\ &= \sum_{k=1}^{r} k\, t_k - p_{(1)} \\ &= \sum_{k=2}^{r} (k-1)\, t_k + \sum_{k=1}^{r} t_k - p_{(1)}. \end{aligned}$$

With jobs scheduled such that $C_i = d_i$ for $i = 1,\ldots,r$, the entire time interval from $d_{(1)} - p_{(1)}$ until $d_{(r)}$ has at least one job scheduled (i.e., there is no idle time). Thus, we have $\sum_{k=1}^{r} t_k - p_{(1)} \geq d_{(r)} - d_{(1)}$ for any jobs (1) and (r) in J^*.

Therefore, $p_{(2)} + \ldots + p_{(r)} - (d_{(r)} - d_{(1)}) \geq \sum_{k=2}^{r} (k-1)\, t_k$.

Since the above is true for every sequence of r jobs, the theorem holds. ■

>> *Insert Figure 1 here* <<

The above property is described pictorially in Figure 1, where the jobs are partitioned into two subsets. Conceptually, Theorem 1 says that jobs must be shifted far enough forward or backward in time from their respective due dates to satisfy the constraint of at most one job on the machine at a given time. The lower bound gives the minimal amount of shifting to accomplish this under the assumption that each subset of jobs can be considered separately.

In general, shifting in one subset will affect other subsets, so that the given bound may not be achievable.

The next two propositions provide dominance rules for a pair of adjacent jobs. We show that when two adjacent jobs are scheduled earlier than a particular time, or after a certain point in time, one job should precede the other in an optimal sequence when the processing times of the jobs satsify certain conditions.

Proposition 3. *If two adjacent jobs, i and j, are scheduled to be completed no later than* min $\{d_i, d_j\}$, *job i should precede job j when* $p_i > p_j$, *and job j should precede job i when* $p_i < p_j$.

Proof. Since both jobs are early, the total earliness for the two jobs can be obtained easily. Let t_1 be the completion time of the later of the two jobs in an arbitrary sequence. Define $TE(x,y)$ as in the proof of Proposition 2. Then, $TE(i,j) = d_j - t_1 + d_i - (t_1 - p_j)$ and $TE(j,i) = d_j - (t_1 - p_i) + d_i - t_1$. Since $TE(i,j) - TE(j,i) = p_j - p_i$, the proposition follows. ■

Proposition 4. *If two adjacent jobs, i and j, cannot be started earlier than* max $\{d_i, d_j\}$ + min $\{p_i, p_j\}$, *job i should precede job j when* $p_i < p_j$, *and job j should precede job i when* $p_i > p_j$.

Proof. In this case, both jobs are tardy. Let t_2 be the starting time of jobs i and j in an arbitrary sequence. Then, $TE(i,j) = t_2 + p_i - d_i + t_2 + p_i + p_j - d_j$, and $TE(j,i) = t_2 + p_j + p_i - d_i + t_2 + p_j - d_j$. Since $TE(i,j) - TE(j,i) = p_i - p_j$, the proposition follows. ■

4. A Branch and Bound Algorithm

We now develop a branch and bound (B&B) algorithm using the properties derived in the previous section. We construct the schedule starting at the end of the sequence. Thus, a node at the k-th level in the B&B tree corresponds to a sequence (and the associated subproblem) for the last k jobs in the sequence. We refer to a sequence of a subset of jobs as

a partial sequence. Branching adds a job to the last unassigned position in a partial sequence.

For each of the nodes generated in the algorithm, a lower bound on the sum of tardiness and earliness of all jobs is computed. Nodes with lower bounds that are greater than the current incumbent solution (or upper bound on an optimal solution) are removed from further consideration (fathomed). The initial upper bound can be set to a very large value or can be obtained using a heuristic. Several heuristics are described in Section 5.

We implemented two different lower bounding schemes. The first provides a tighter bound, but is more complicated to compute. We first describe the more complicated bound (which we call Bound A), and then discuss the simpler bound (referred to as Bound B), which is actually a lower bound on Bound A. Conceptually, both bounds are based on a partition of the jobs into three mutually exclusive and collectively exhaustive subsets, and a lower bound on tardiness and earliness is computed for each of these subsets. The three subsets are: (1) the set of jobs in the partial sequence; (2) the set of unscheduled jobs with due dates greater than the starting time of the partial sequence; and (3) all other jobs.

Notation:

σ = partial sequence

s = optimal starting time for σ under the constraint that all jobs not in σ can be completed by time s

$G_\sigma(t)$ = marginal increase in the total tardiness and earliness for σ if its starting time is delayed by one unit from t, $t \geq s$. $G_\sigma(t)$ can be computed as the number of jobs in σ that are not early minus the number of early jobs in σ that precede the first idle period within the optimal schedule for σ starting at time t.

$\tau_\sigma(t)$ = set of jobs not in σ with due dates greater than t

$p_{<k>}(t)$ = processing time of the k-th shortest job in $\tau_\sigma(t)$

d_i = due date of job i

$d_{[i]}$ = due date of job with the i-th latest due date among jobs not in σ

We first make some observations that will be useful in our analysis.

1. $G_\sigma(t)$ is a non-negative, non-decreasing step function. It increases at values of t for which the resulting optimal scheduling (starting at time t) has the property that (at least) one job is early in an optimal schedule for σ starting at $t-\varepsilon$ and the same job(s) is (are) tardy in an optimal schedule starting at $t+\varepsilon$ for all $\varepsilon > 0$. Since $G_\sigma(t)$ is a non-negative, non-decreasing step function, $\int_s^t G_\sigma(u)\,du$, which is the cost of delaying σ from s until time t, is a non-decreasing, piecewise linear, convex function of t (for fixed s).
2. $|\tau_\sigma(t)|$ is non-increasing with t and is also a step function.

Bound A

If σ were considered alone, we could obtain the (unconstrained) optimal value of the total tardiness and earliness by using the optimal timing algorithm mentioned earlier. However, since jobs not in σ must be completed prior to σ, the optimal *constrained* starting time of σ is the maximum of the optimal unconstrained starting time and the sum of the processing times of jobs not in σ. In either case, the corresponding objective value is easy to compute.

Let us now consider the jobs in $\tau_\sigma(s)$. These jobs induce additional tardiness and earliness for two reasons. First, when these jobs are included, it may be optimal to delay the starting time of σ past s. Second, even if σ is delayed, some of the jobs not in σ may be early, since they have to be scheduled prior to σ. The penalty due to the first effect can be computed exactly (for any delay) using $G_\sigma(t)$. Computing the penalty due to the second effect requires the specification of the entire sequence for $\tau_\sigma(s)$, which we do not have at this stage. We can, however, compute a lower bound on it. The sum of the two penalties provides a lower bound on the additional tardiness and earliness induced by $\tau_\sigma(s)$, *given a particular starting time for* σ. If we optimize this expression with respect to the starting time of σ, we will have a lower bound on the additional tardiness and earliness induced by jobs in $\tau_\sigma(s)$.

We first derive a lower bound on the earliness of the jobs in $\tau_\sigma(t)$ for some arbitrary t, assuming that σ starts at time t. The lower bound is obtained by assuming that all of the jobs in $\tau_\sigma(t)$ are performed sequentially, with the last of them being completed at time t. In other words, no other jobs that might cause additional earliness are interspersed among the the jobs in $\tau_\sigma(t)$. If other jobs were so scheduled, the penalty would be even larger; thus we have a lower bound. Under this condition, the minimum earliness is obtained by sequencing the jobs in $\tau_\sigma(t)$ using LPT (longest processing time first), and the resulting earliness is

$$\sum_{i \in \tau_\sigma(t)} (d_i - t) + \sum_{k=1}^{|\tau_\sigma(t)|} (|\tau_\sigma(t)| - k) p_{<k>}(t) . \qquad (1)$$

The first term in the above expression is the earliness incurred after time t, and the second term is the earliness incurred before time t.

The cost associated with the delay of σ until time t is

$$\int_s^t G_\sigma(u)\, du . \qquad (2)$$

The first term in (1) is monotonically non-increasing with t and it is easy to show that it is a piecewise linear, convex non-increasing function with changes in the slope at values of t equal to the due date of a job in $\tau_\sigma(t)$. The second term in (1) is a step function that changes value only at values of t equal to d_i for some i, and with some algebra, it can be shown that it is monotonically non-increasing with t. The term in (2) is piecewise linear, convex, and non-decreasing with t. Since all three terms are piecewise linear (although not necessarily continuous) functions of t, the value of t that minimizes their sum must be either a value at which the slope of one (or more) of the functions changes, or one of the points of discontinuity in the step function. This implies that the set of dominant solutions is composed of: (i) due dates of jobs in $\tau_\sigma(s)$; and (ii) values of t where the slope of the term in (2) changes. The set of values in (i) has cardinality less than the total number of jobs. Since $G_\sigma(\cdot)$ is an integer less than or equal to the number of jobs (from the definition of $G_\sigma(\cdot)$), the set of values in (ii) also has cardinality less than the number of jobs. Thus, the number of values of t that must be evaluated is less than $2n$. The lowest cost alternative provides the lower bound.

Finally, we consider jobs neither in σ nor in $\tau_\sigma(\cdot)$. We use the results of Theorem 1 to obtain a lower bound on tardiness and earliness for this subset of jobs. The lower bound for all jobs is computed as the sum of the lower bounds for the three subsets.

Bound B

Bound B is

$$\sum_{i=1}^{G_\sigma(s)} (d_{[i]} - s)^+ \tag{3}$$

In the Appendix, we show that this is a lower bound on Bound A. In the proof, we show that the expression in (3) is always less than or equal to the minimum (with respect to t) of the sum of the first term in (1) and the expression in (2). Since Bound A also includes the second term in (1), which is non-negative, it follows that Bound B is always greater than or equal to Bound B.

We use a *depth-first* branching rule, i.e., the node with the most jobs in the corresponding partial sequence is selected for branching. Ties are broken in favor of the node with the minimum lower bound. We use the results of Proposition 4 to avoid constructing sequences that are dominated.

5. Heuristic Algorithms

The branch and bound algorithm described in the previous section is not computationally viable for large problems. Therefore, we also develop several heuristic algorithms for sequencing, but continue to use the DP algorithm mentioned earlier to determine the optimal timing for a given sequence. Some of these heuristic algorithms use properties of the optimal solution described earlier in the paper, while others first construct a sequence through job insertions and improve it using various interchange methods. Some of the interchange methods previously used for related scheduling problems include simple

search methods, taboo search, and simulated annealing. We test several heuristic algorithms based on some of these approaches, as described below.

1) EDD (Earliest due date)

Jobs are sorted in increasing (non-decreasing) order of due dates. Ties are broken using the earliest starting time rule (see below).

2) EST (Earliest starting time)

Jobs are sorted in increasing (non-decreasing) order of their desired starting times, i.e., due date minus processing time ($d_i - p_i$). Ties are broken using the earliest due date rule.

3) PREC (Precedence relationships)

This procedure is based on an analysis of all pairs of jobs using the results of Proposition 2. Priorities (denoted by k_i, i=1,...,n) of the jobs are computed as follows. Starting with k_i=0 for i=1,...,n, we update the k_is by adding 1 to k_i and subtracting 1 from k_j if job i should precede job j, where these precedence relationships are determined by applying the results of Proposition 2 under the assumption that the two jobs are adjacent and that their starting times are not restricted. After comparing all pairs of jobs, a sequence is obtained by sorting the jobs in decreasing (non-increasing) order of the k_i values. Ties are broken by the EDD rule.

4) NEH (Algorithm of Nawaz, Enscore, and Ham)

This is an adaptation of the algorithm suggested in Nawaz *et al.* (1983) for permutation flowshops (called the NEH algorithm in other research). In the original NEH algorithm, jobs are sorted in decreasing (non-increasing) order of processing times, then a final solution is obtained by considering each job in the specified order and inserting it into the sequence in the position that results in the best partial solution. Since due dates are considered in our problem, pre-sorting the jobs in decreasing order of

processing times may not be appropriate. We evaluate four methods for sorting the jobs: EDD (earliest due date), LDD (latest due date), SPT (shortest processing time), and LPT (longest processing time). NEHedd, NEHldd, NEHspt, and NEHlpt denote the NEH algorithm along with the sorting method. See Nawaz *et al.* for details on the original NEH algorithm.

5) API (Adjacent pair interchange)

This algorithm starts with an initial solution from one of the previous heuristics and iteratively considers interchanging adjacent jobs, implementing the change whenever there is an improvement.

6) PI (Pairwise interchange)

This algorithm starts with a complete sequence (in our experiments, we use the one from API) and tries to improve the solution by considering all possible pairwise interchanges. Unlike API, the jobs need not be adjacent. Consequently, at any point in the procedure, there are $n(n-1)/2$ possible interchanges from which to choose, where n is the number of jobs. (We refer to all sequences that can be obtained through one such pairwise interchange as a 'neighboring solution.') If the best interchange results in an improvement, it is implemented, and the process is repeated. The algorithm terminates when no further improvement can be achieved.

7) TS (Taboo search method)

This procedure is an implementation of taboo (tabu) search (see Widmer and Hertz 1989, Taillard 1990, and Glover 1990, among others for additional details on taboo search). In our implementation, we allow interchanges to be made to the best neighboring solution even if it is worse than the current solution. However, a list L of taboo moves (moves that are not allowed at the present iteration) is maintained and updated when a move is made. We add taboo moves to the list as follows. If job x is in

the i-th position in the sequence and y is in the j-th position before a move and the two jobs are interchanged, we do not allow jobs x and y to return to positions i and j, respectively, for a specified number of iterations. Whenever an interchange is made, these two (types of) taboo moves are added to L, and the two oldest (types of) taboo moves in L are removed. The algorithm terminates when no improvement has occurred for a specified number of iterations (say I_{max}). Note that the number of taboo moves ($|L|$) and I_{max} affect the solution quality and computation time of the algorithm. After some experimentation, we set $|L| = 7$ and $I_{max} = \min(n,15)$, where n is the number of jobs.

Because of the difficulties associated with setting various parameters and defining search regions, we decided not to use simulated annealing in our experiments. Some research has suggested that the quality of the solution from simulated annealing may be sensitive to these particular choices (e.g., Osman and Potts 1989 and Potts and Van Wassenhove 1991). Our results in the next section suggest that the simpler procedures described above may be more than adequate for finding a very good solution.

6. Computational Experiments

6.1. Test of the heuristics

We first test the heuristic algorithms described in Section 5 to determine which would consistently provide the best initial incumbent solution for the B&B algorithm. For this purpose, we randomly generated 1000 problems. The number of jobs ranges from 20 to 80. Processing times and due dates of the jobs are generated following the method by Potts and Van Wassenhove (1982), which is a popular method to generate single-machine scheduling problems with due dates. The processing times are generated from a discrete uniform distribution with a range of 1 to 30. The due dates are generated from a discrete uniform distribution $[P(1-T-R/2), P(1-T+R/2)]$, where P is the sum of the processing times

of all operations, R is the due-date range and T is the tardiness factor. T ranges from 0.1 to 0.5 and R ranges from 0.8 to 1.8 in our experiments. Combinations of T and R values are selected so that the due dates are nonnegative. (Refer to Table 5 for the combinations.) The algorithms were coded in FORTRAN and run on a personal computer with an 80486 processor.

It was not possible to find optimal solutions for all problems. Consequently, we decided to use a *relative performance ratio* (RPR), defined as the ratio of the solution value from an algorithm to the best known solution value for the problem, to measure solution quality. In this set of problems, TS always produced the best solution, although this is not surprising since it starts with the best of the solutions from the other heuristics, and then executes a computationally intensive improvement routine. Table 1 shows the mean, standard deviation and the maximum value of RPR for each algorithm. It also shows the number of problems for which each algorithm found the best solution. The mean CPU times are also given for different problem sizes (where the number of jobs in the problems is denoted by n). For the improvement algorithms (API, PI, and TS) we report the additional (net) CPU time after obtaining an initial sequence. In the experiments, API uses the best solution from the insertion heuristics (NEH) as an initial sequence, PI uses the solution from API, and TS uses the solution from PI as the initial sequence. Hence, the actual CPU time for TS is the sum of CPU times for NEH, API, PI and TS.

>> *Insert Table 1 here* <<

The insertion heuristics generally outperformed the simple sorting algorithms (EDD, EST, and PREC), but require much more CPU time than the sorting algorithms. It is instructive to point out that PREC, which is the only heuristic based on the various dominance properties, did not outperform the simpler sorting algorithms. The adjacent pair interchange algorithm (API) provided noticeable improvements with very little additional CPU time. Smaller incremental improvements were obtained from PI and TS, but the

additional computing time was substantial in the case of TS. Considering computation time and solution quality, API or PI might serve as a good heuristic.

6.2. Experiments with the branch and bound algorithm

The primary purpose of the experiments in this section is to ascertain which types of problems can be solved optimally, and to determine the effectiveness of the dominance rules and bounding schemes. We randomly generated a total of 400 problems using a method similar to the one used in §6.1. The number of jobs ranges from 10 to 28. In half of the problems, the processing times of jobs range from 1 to 10, while in the other half they range from 1 to 100. Twenty different combinations of T and R values (refer to Table 5) are used.

The solution from PI is used as the initial incumbent solution, since this heuristic produced good solutions in a short period of time. The B&B algorithm was coded in FORTRAN and run on a personal computer with an 80486 processor. A 3600-second (one hour) CPU time trap was used for each problem. Since the depth-first rule was used for node selection, the memory requirements did not exceed the capacity of the computer memory (640KB). (Indeed, we selected the depth-first rule to avoid problems in this regard.)

Table 2 shows the results for various problem parameters. The reported statistics include number of problems solved within one hour, mean and median CPU times, and mean and median percentages of subproblems considered (100 times the number of nodes generated by the algorithm divided the maximum number of nodes that might have been generated without fathoming). We did not compute the mean CPU time if all 20 problems in a group could not be solved within the time limit, but we do report the median among those that were solved.

>> *Insert Table 2 here* <<

As anticipated, the CPU time increases quickly as the problem size increases, but not as quickly as the number of different sequences. The percentage of nodes evaluated

decreases as the number of jobs increases, which means the bounding schemes are effective. It is interesting to note that the tighter bound (Bound A) performs similarly to the simpler bound (Bound B). The reason for this is that Bound A requires significantly more computation time (because of the small optimization problem that must be solved to determine each bound), but does not appear to produce a sufficiently tighter bound to have much of an impact on either the number of nodes considered or the overall CPU time. Thus, while, in principle, the tighter bound should be better, in this case, the benefit appears to be negligible.

The effects of different tardiness factors and due-date ranges are shown in Tables 3, 4, and 5. Due-date ranges significantly affect the performance of the algorithm. The reason seems to be that optimal solution values tend to be small in problems in which due dates are widely distributed, so the initial upper bound may be small as well, which allows us to to fathom nodes earlier. In the extreme case where the value of the upper bound is zero, the algorithm can be terminated immediately. On the other hand, when the due-date range is small, the upper bound is usually large. In this case, a partial sequence with a small number of jobs rarely has a lower bound greater than the upper bound even when the partial sequence is quite poor. Only after most of the jobs have been sequenced is it possible to fathom nodes. This explains the results in Table 3.

>> *Insert Tables 3, 4 , and 5 here* <<

The tardiness factor has a similar effect on the performance of the algorithm, as problems with larger tardiness factors result in larger solution values. However, the effect of the tardiness factor was not as significant as that of the due-date range. This is because the lower bound for jobs in a partial sequence becomes tighter as the tardiness factor increases.

We now turn to a comparison of the heuristics and the B&B algorithm using the same 400 test problems. The CPU times for the algorithms were given in Table 2. Table 6 shows relative performance ratios of the two best-performing heuristics, PI and TS.

>> *Insert Table 6 here* <<

The solutions from the heuristics are nearly as good as those from the B&B algorithm. The objective values of the solutions of PI and TS were, on average, only 0.49% and 0.14% greater, respectively, than the best solutions. Of the 343 problems that could be solved optimally using one of the bounding approaches, PI identified optimal solutions in 296 cases, while TS did so for 319 problems. Moreover, in 11 problems TS found a better solution than the B&B algorithm with a CPU time trap. Although TS requires much more CPU time than other heuristics tested here, the computational requirements are still quite small in comparison with the time needed for the B&B algorithm. In further computational tests (with another B&B algorithm that uses TS to obtain an initial solution), however, we found that there was little overall benefit from using TS (rather than PI) as an initial solution for the B&B algorithm.

7. Concluding Remarks

We have identified several properties of optimal solutions for the single machine scheduling problem with the objective of minimizing mean tardiness and earliness. We have developed both a branch-and-bound algorithm and a heuristic algorithm (PREC) based on these properties. We have also tested a variety of other heuristic procedures.

Computational results suggest that problems with up to 20 jobs can be solved optimally within a modest amount of CPU time. We observed that the CPU time requirements depend heavily on the range of due dates. That is, problems with less widely dispersed due dates were more difficult to solve optimally with the B&B algorithm. Additional research is needed to develop more powerful dominance criteria and tighter (or faster-to-compute) lower bounds for problems of this type.

The results also indicate that while some simple sorting heuristics (including PREC) are able to produce solutions within roughly 30% of optimality, a large portion of this gap can be reduced through the use of improvement routines involving pairwise interchange. Although some of these improvement routines can be computationally intensive, the computation requirements are much less than for the optimal solution procedure.

Acknowledgement

Candace A. Yano was supported in part by National Science Foundation Grant EID-9023674 to the University of Michigan.

Appendix

Proof that Bound B is a Lower Bound on Bound A

We start with the expression for Bound A and show that the sum of two of its three non-negative terms is greater than or equal to Bound B. More precisely, we show that

$$\sum_{i \in \tau_\sigma(t)} (d_i - t) + \int_s^t G_\sigma(u)\, du \geq \sum_{i=1}^{G_\sigma(s)} (d_{[i]} - s)^+, \qquad \text{(A-1)}$$

for all $t \geq s$.

Note that the left hand side of (A-1) can be rewritten as

$$\sum_{i \in \tau_\sigma(t)} d_i - t\,|\tau_\sigma(t)| + \int_s^t G_\sigma(u)\, du \geq \sum_{i \in \tau_\sigma(t)} d_i - t\,|\tau_\sigma(t)| + (t-s)\,G_\sigma(s) \qquad \text{(A-2)}$$

since $G_\sigma(t)$ is non-decreasing in t for $t \geq s$.

We will show that

$$\sum_{i \in \tau_\sigma(t)} d_i - t\,|\tau_\sigma(t)| + (t-s)\,G_\sigma(s) \geq \sum_{i=1}^{G_\sigma(s)} (d_{[i]} - s)^+. \qquad \text{(A-3)}$$

Since $|\tau_\sigma(t)|$ is decreasing with t, we know that $|\tau_\sigma(t)| \leq |\tau_\sigma(s)|$. We prove the inequality in each of the following three cases.

Case 1: $G_\sigma(s) < |\tau_\sigma(t)| \leq |\tau_\sigma(s)|$

Since all jobs in $\tau_\sigma(s)$ have due dates greater than s, the jobs in $\tau_\sigma(s)$ with the $G_\sigma(s)$ ($< |\tau_\sigma(s)|$) largest due dates all have due dates greater than s. Consequently, we can ignore the positive sign on the right hand side and rewrite (A-3) as

$$\sum_{i \in \tau_\sigma(t)} d_i - t\,|\tau_\sigma(t)| + (t-s)\,G_\sigma(s) \geq \sum_{i=1}^{G_\sigma(s)} d_{[i]} - s\,G_\sigma(s)$$

or

$$\sum_{i \in \tau_\sigma(t)} d_i - \sum_{i=1}^{G_\sigma(s)} d_{[i]} \geq t\,(|\tau_\sigma(t)| - G_\sigma(s)) \qquad \text{(A-4)}$$

Since $|\tau_\sigma(t)| > G_\sigma(s)$, $\tau_\sigma(t)$ is a superset of the set composed of the $G_\sigma(s)$ jobs with the largest due dates in $\tau_\sigma(s)$. Thus, the left hand side of the above expression can be written as

$$\sum_{i=G_\sigma(s)+1}^{|\tau_\sigma(t)|} d_{[i]}, \qquad \text{(A-5)}$$

which is simply the sum of due dates for jobs in $\tau_\sigma(t)$ but not in the intersection of the two sets. Since the jobs are in $\tau_\sigma(t)$, they have due dates greater than t. Then, since there are $|\tau_\sigma(t)| - G_\sigma(s)$ such due dates in the sum in (A-5), the sum is greater than $t(|\tau_\sigma(t)| - G_\sigma(s))$.

Case 2: $|\tau_\sigma(t)| \leq G_\sigma(s) < |\tau_\sigma(s)|$

In this case, the transformation from (A-3) to (A-4) is still valid. Since $|\tau_\sigma(t)| \leq G_\sigma(s)$, $\tau_\sigma(t)$ is a subset of the set composed of the $G_\sigma(s)$ ($< |\tau_\sigma(s)|$) jobs with the largest due dates in $\tau_\sigma(s)$. Thus, the left hand side of (A-4) can be written as

$$-\sum_{i=|\tau_\sigma(t)|+1}^{G_\sigma(s)} d_{[i]},$$

which is simply the additive inverse of the sum of due dates for jobs in the larger set but not in the smaller set. Reversing the direction of the inequality (because both sides of (A-4) are negative), we can rewrite (A-4) as

$$\sum_{i=|\tau_\sigma(t)|+1}^{G_\sigma(s)} d_{[i]} \leq t\,(G_\sigma(s) - |\tau_\sigma(t)|),$$

which we need to prove.

Since the jobs whose due dates are in the sum on the left hand side are not in $\tau_\sigma(t)$, they have due dates less than t. Then, since there are exactly $G_\sigma(s) - |\tau_\sigma(t)|$ such due dates in the sum, the sum is less than or equal to $t\,(G_\sigma(s) - |\tau_\sigma(t)|)$.

Case 3: $|\tau_\sigma(t)| \le |\tau_\sigma(s)| \le G_\sigma(s)$

Since $d_{[i]} \le s$ for $i \notin \tau_\sigma(s)$, in this case we can rewrite the right hand side of (A-3) as

$$\sum_{i=1}^{|\tau_\sigma(s)|} (d_{[i]} - s) = \sum_{i=1}^{|\tau_\sigma(s)|} d_{[i]} - s\,|\tau_\sigma(s)|$$

and (A-3) then becomes

$$\sum_{i \in \tau_\sigma(t)} d_i - \sum_{i=1}^{|\tau_\sigma(s)|} d_{[i]} \ge t\,(|\tau_\sigma(t)| - G_\sigma(s)) + s\,(G_\sigma(s) - |\tau_\sigma(s)|),$$

which can be rewritten (reversing the direction of the inequality) as

$$\sum_{i \in \tau_\sigma(s) \setminus \tau_\sigma(t)} d_i \le t\,(|\tau_\sigma(s)| - |\tau_\sigma(t)|) + (t-s)\,(G_\sigma(s) - |\tau_\sigma(s)|),$$

where $A \setminus B$ denotes the set composed of elements in set A but not in set B.

Each of the due dates on the left hand side of the above expression must be between s and t and therefore less than t. Moreover, there are exactly $|\tau_\sigma(s)| - |\tau_\sigma(t)|$ such terms. Thus, the left hand side of the above expression is less than the first term on the right hand side. Since the second term on the right hand side is non-negative, the inequality is satisfied. ■

References

Abdul-Razaq, T. S. and C. N. Potts, 1988. Dynamic Programming State-Space Relaxation for Single-Machine Scheduling. *Journal of Operational Research Society*, Vol.39, No.2, pp.141-152.

Ahmed, M. U. and P. S. Sundararaghavan, 1990. Minimizing the Weighted Sum of Late and Early Completion Penalties in a Single Machine. *IIE Transactions*, Vol.22, No.3, pp.288-290.

Bagchi, U., Y. L. Chang, and R. S. Sullivan, 1987a. Minimizing Absolute and Squared Deviations of Completion Times with Different Earliness and Tardiness Penalties and a Common Due Date. *Naval Research Logistics Quarterly*, Vol.34, pp.739-751.

Bagchi, U., R. S. Sullivan, and Y. L. Chang, 1986. Minimizing Mean Absolute Deviation of Completion Times about a Common Due Date. *Naval Research Logistics Quarterly*, Vol.33, pp.227-240.

Bagchi, U., R. S. Sullivan, and Y. L. Chang, 1987b. Minimizing Mean Squared Deviation of Completion Times about a Common Due Date. *Management Science*, Vol.33, No.7, pp.894-906.

Baker, K. R. and G. D. Scudder, 1990. Sequencing with Earliness and Tardiness Penalties: A Review. *Operations Research*, Vol.38, No.1, pp.22-36.

Cheng, T. C. E., 1987. An Algorithm for the CON Due-Date Determination and Sequencing Problem. *Computers and Operations Research*, Vol.14, No.6, pp.537-542.

De, P., J. B. Ghosh, and C. E. Wells, 1991. Scheduling to Minimize Weighted Earliness and Tardiness about a Common Due-Date. *Computers and Operations Research*, Vol.18, No.5, pp.465-475.

Ferris, M. C. and M Vlach, 1992. Scheduling with Earliness and Tardiness Penalties. *Naval Research Logistics*, Vol.39, pp.229-245.

Fry, T. D., R. D. Armstrong, and J. H. Blackstone, 1987a. Minimizing Weighted Absolute Deviation in Single Machine Scheduling. *IIE Transactions*, Vol.19, No.4, pp.445-450.

Fry, T. D., G. K. Leong, and T. R. Rakes, 1987b. Single Machine Scheduling: A Comparison of Two Solution Procedures. *OMEGA The International Journal of Management Science*, Vol.15, No.4, pp.277-282.

Garey, M. R., R. E. Tarjan, and G. T. Wilfong, 1988. One-Processor Scheduling with Symmetric Earliness and Tardiness Penalties. *Mathematics of Operations Research*, Vol.13, No.2, pp.330-348.

Glover, F., 1990. Tabu Search: A Tutorial. *Interfaces*, Vol.10, No.4, pp.74-94.

Gupta, S. K. and T. Sen, 1983. Minimizing a Quadratic Function of Job Lateness on a Single Machine. *Engineering Costs and Production Economics*, Vol.7, pp.187-194.

Hall, N. G. and M. E. Posner, 1991. Earliness-Tardiness Scheduling Problems, I: Weighted Deviation of Completion Times about a Common Due Date. *Operations Research*, Vol.39, No.5, pp.836-846.

Hall, N. G., W. Kubiak, and S. P. Sethi, 1991. Earliness-Tardiness Scheduling Problems, II: Deviation of Completion Times about a Restrictive Common Due Date. *Operations Research*, Vol.39, No.5, pp.847-856.

Kanet, J. J., 1981. Minimizing the Average Deviation of Job Completion Times about a Common Due Date. *Naval Research Logistics Quarterly*, Vol.28, No.4, pp.643-651.

Lakshminarayan, S., R. Lakshmanan, R. L. Papineau, and R. Rochette, 1978. Optimal Single-Machine Scheduling with Earliness and Tardiness Penalties. *Operations Research*, Vol.26, No.4, pp.1079-1082.

Lee, C-Y., S. L. Danusaputro, and C-S. Lin, 1991. Minimizing Weighted Number of Tardy Jobs and Weighted Earliness-Tardiness Penalties about a Common Due date. *Computers and Operations Research*, Vol.18, No.4, pp.379-389.

Nawaz, M., E. E. Enscore Jr., and I. Ham, 1983. A Heuristic Algorithm for the m-Machine, n-Job Flow-shop Sequencing Problem. *OMEGA The International Journal of Management Science*, Vol.11, pp.91-95.

Osman, I. H. and C. N. Potts, 1989. Simulated Annealing for Permutation Flow-Shop Scheduling. *OMEGA The International Journal of Management Science*, Vol.17, pp.551-557.

Ow, P. S. and T. E. Morton, 1989. The Single Machine Early/Tardy Problem. *Management Science*, Vol.35, No.2, pp.177-191.

Potts, C. N. and L. N. Van Wassenhove, 1982. A Decomposition Algorithm for the Single Machine Total Tardiness Problem. *Operations Research Letters*, Vol.1, No.5, pp.177-181.

Potts, C. N. and L. N. Van Wassenhove, 1991. Single Machine Tardiness Sequencing Heuristics. *IIE Transactions*, Vol.23, No.4, pp.346-354.

Seidmann, A., S. S. Panwalkar, and M. L. Smith, 1981. Optimal Assignment of Due-Dates for a Single Processor Scheduling Problem. *International Journal of Production Research*, Vol.19, No.4, pp.393-399.

Sidney, J. B., 1977. Optimal Single-Machine Scheduling with Earliness and Tardiness Penalties. *Operations Research*, Vol.25, No.1, pp.62-69.

Sundararaghavan, P. S. and M. U. Ahmed, 1984. Minimizing the Sum of Absolute Lateness in Single-Machine and Multimachine Scheduling. *Naval Research Logistics Quarterly*, Vol.31, pp.325-333.

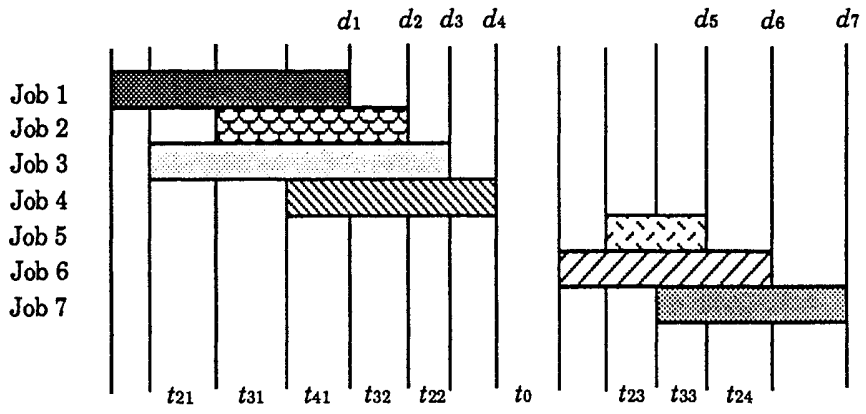
Szwarc, W., 1989. Single-Machine Scheduling to Minimize Absolute Deviation of Completion Times from a Common Due Date. *Naval Research Logistics*, Vol.36, pp.663-673.

Taillard, E., 1990. Some Efficient Heuristic Methods for the Flow Shop Sequencing Problem. *European Journal of Operational Research*, Vol.47, pp.65-74.

Widmer, M. and A. Hertz, 1989. A New Heuristic Method for the Flow Shop Sequencing Problem. *European Journal of Operational Research*, Vol.41, pp.186-193.

Yano, C. A. and Y-D. Kim, 1986. Algorithms for Single Machine Scheduling Problems Minimizing Tardiness and Earliness. Technical Report 86-40, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, MI 48109-2117, USA.

Yano, C. A. and Y-D. Kim, 1991. Algorithms for a Class of Single-Machine Tardiness and Earliness Problems. *European Journal of Operational Research*, Vol.52, No.2, pp.167-178.



Lower bound = (t_{21} + t_{22} + t_{23} +t_{24}) + 2 (t_{31} +t_{32} + t_{33}) + 3 t_{41}

Figure 1. Lower bound on earliness and tardiness of jobs not yet sequenced

Table 1. Test results for heuristic algorithms

algorithms	solution quality				average CPU time (seconds)			
	# times † best	relative performance ratio						
		mean	st. dev.	maximum	n=20	n=40	n=60	n=80
EDD	8	1.340	0.164	1.907	0.00	0.00	0.01	0.01
EST	3	1.349	0.168	2.033	0.00	0.00	0.01	0.01
PREC	6	1.314	0.174	1.952	0.00	0.01	0.02	0.02
NEHedd	52	1.281	0.269	2.760	0.08	0.67	2.46	6.22
NEHldd	47	1.355	0.344	3.480	0.08	0.68	2.46	6.17
NEHspt	79	1.088	0.084	1.605	0.07	0.54	1.85	4.44
NEHlpt	101	1.109	0.093	1.491	0.08	0.65	2.41	6.06
API	410	1.021	0.029	1.158	0.01	0.06	0.15	0.28
PI	757	1.003	0.010	1.109	0.12	2.87	20.17	78.89
TS	1000	1.000	0.000	1.000	1.61	15.86	61.42	170.24

† The number of problems in which the corresponding algorithm found the best solution.

Table 2. Peformance of the B&B algorithm for different problem sizes

number of jobs	range of processing times	bound used	no. of problems solved	% of subproblems considered		CPU time (seconds)		mean CPU time for heuristics (sec.)	
				mean	median	mean	median	PI	TS
10	[1,10]	A	20	0.00752	0.00365	.19	.10	0.04	0.13
		B	20	0.00876	0.00381	.15	.06		
10	[1,100]	A	20	0.00548	0.00333	.14	.10	0.03	0.15
		B	20	0.00689	0.00333	.12	.06		
12	[1,10]	A	20	1.77E–4	2.97E–5	.62	.14	0.05	0.28
		B	20	2.57E–4	2.97E–5	.56	.11		
12	[1,100]	A	20	2.03E–4	4.95E–5	.67	.19	0.06	0.30
		B	20	2.39E–4	5.24E–5	.51	.15		
14	[1,10]	A	20	2.19E–6	4.27E–7	1.31	.31	0.08	0.53
		B	20	3.98E–6	5.86E–7	1.46	.27		
14	[1,100]	A	20	1.52E–5	1.57E–6	10.03	.81	0.10	0.57
		B	20	2.52E–5	1.58E–6	10.13	.61		
16	[1,10]	A	20	1.25E–7	2.92E–9	11.99	.41	0.14	0.87
		B	20	1.28E–7	2.98E–9	9.19	.36		
16	[1,100]	A	20	5.10E–8	7.26E–9	8.15	.98	0.14	0.96
		B	20	7.11E–8	7.31E–9	7.37	.65		
18	[1,10]	A	20	3.18E–9	3.29E–11	158.14	1.31	0.20	1.32
		B	20	4.64E–9	4.47E–11	140.15	1.20		
18	[1,100]	A	20	7.48E–10	5.51E–11	43.72	1.80	0.20	1.32
		B	20	1.06E–9	5.55E–11	35.33	1.24		
20	[1,10]	A	18	†	4.86E–13	†	7.20	0.26	1.94
		B	18	†	5.14E–13	†	5.07		
20	[1,100]	A	17	†	9.94E–14	†	1.48	0.31	1.89
		B	17	†	9.97E–14	†	1.05		
22	[1,10]	A	16	†	5.59E–15	†	38.90	0.34	2.38
		B	16	†	5.94E–15	†	24.74		
22	[1,100]	A	15	†	4.17E–15	†	27.95	0.43	2.74
		B	16	†	4.59E–15	†	19.06		
24	[1,10]	A	13	†	2.46E–17	†	106.03	0.46	3.02
		B	12	†	4.63E–17	†	116.45		
24	[1,100]	A	12	†	1.89E–17	†	67.43	0.70	3.54
		B	12	†	2.67E–17	†	58.64		
26	[1,10]	A	12	†	7.29E–20	†	186.62	0.72	4.23
		B	12	†	1.07E–19	†	170.01		
26	[1,100]	A	15	†	1.21E–19	†	286.85	0.82	4.69
		B	14	†	1.26E–19	†	195.37		
28	[1,10]	A	11	†	2.36E–22	†	410.33	0.96	5.19
		B	11	†	2.95E–22	†	331.50		
28	[1,100]	A	12	†	7.00E–23	†	130.46	0.96	5.52
		B	12	†	9.41E–23	†	116.12		

† Mean values were not computed for these entries because some problems were not solved to optimality.

Table 3. Peformance of the B&B algorithm for different due-date ranges (R)

due-date range (R)	no. of problems tested	Bound A			Bound B		
		no. of probs. solved	median CPU time (seconds)	median % of subproblems considered	no. of probs. solved	median CPU time (seconds)	median % of subproblems considered
0.8	100	61	115.66	1.35E–10	60	109.70	2.34E–10
1.0	100	83	13.99	3.29E–11	84	12.09	3.77E–11
1.2	80	78	1.04	3.99E–12	79	.79	3.99E–12
1.4	60	59	.57	1.03E–13	59	.47	1.03E–13
1.6	40	39	.38	7.42E–13	39	.38	7.45E–13
1.8	20	20	.31	2.94E–13	20	.20	2.96E–13

Table 4. Peformance of the B&B algorithm for different tardiness factors (T)

Tardiness Factor (T)	no. of problems tested	Bound A			Bound B		
		no. of probs. solved	median CPU time (seconds)	median % of subproblems considered	no. of probs. solved	median CPU time (seconds)	median % of subproblems considered
0.1	120	111	.90	9.88E–13	110	.75	9.88E–13
0.2	100	86	1.07	2.94E–12	87	.83	2.94E–12
0.3	80	68	1.90	5.73E–12	69	1.62	7.54E–12
0.4	60	47	10.72	1.42E–11	47	8.52	1.43E–11
0.5	40	28	139.87	9.78E–11	28	132.83	1.31E–10

Table 5. Peformance of the B&B algorithm for various T and R values

T	R	number of problems tested	Bound A			Bound B		
			no of probs. solved	median CPU time (seconds)	median % of subproblems considered	no of probs. solved	median CPU time (seconds)	median % of subproblems considered
0.1	0.8	20	12	42.98	1.45E–10	11	59.08	2.80E–10
0.1	1.0	20	19	1.33	4.87E–13	19	1.06	5.14E–13
0.1	1.2	20	20	1.17	3.11E–13	20	.91	3.99E–13
0.1	1.4	20	20	.43	1.81E–14	20	.38	1.81E–14
0.1	1.6	20	20	.38	9.94E–14	20	.38	9.97E–14
0.1	1.8	20	20	.31	2.94E–13	20	.20	2.96E–13
0.2	0.8	20	12	230.12	1.26E–10	13	144.54	2.34E–10
0.2	1.0	20	16	14.98	2.55E–12	16	12.09	2.63E–12
0.2	1.2	20	19	.60	6.72E–14	19	.42	7.29E–14
0.2	1.4	20	20	.48	6.15E–14	20	.38	6.15E–14
0.2	1.6	20	19	.33	7.42E–13	19	.33	7.45E–13
0.3	0.8	20	13	62.05	1.00E–10	12	70.41	1.77E–10
0.3	1.0	20	17	13.99	4.43E–11	18	20.10	7.91E–11
0.3	1.2	20	19	.61	1.07E–13	20	.47	1.08E–13
0.3	1.4	20	19	.99	3.76E–13	19	.80	3.89E–13
0.4	0.8	20	12	27.54	1.49E–10	12	17.43	3.16E–10
0.4	1.0	20	15	5.87	3.99E–12	15	3.71	1.08E–11
0.4	1.2	20	20	7.84	3.99E–12	20	5.02	3.99E–12
0.5	0.8	20	12	115.66	9.79E–11	12	109.70	1.31E–10
0.5	1.0	20	16	549.71	8.52E–11	16	344.35	8.67E–11

Table 6. Performance of heuristic algorithms compared with the B&B algorithm (using Bound B)

		PI	TS
(relative) performance ratio	mean	1.00493	1.00142
	maximum	1.10015	1.07547
number of solutions better than best incumbent from B&B		0	11
number of solutions proven to be optimal		296	319
number of solutions worse than best incumbent from B&B		52	25