ALGORITHMS FOR A CLASS OF SINGLE-MACHINE WEIGHTED TARDINESS AND EARLINESS PROBLEMS

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

and

Yeong-Dae KIM
Department of Industrial Engineering
Korean Advanced Institute of Science and Technology
Seoul, Korea

January 1989
Revised September 1989

ALGORITHMS FOR A CLASS OF SINGLE-MACHINE WEIGHTED TARDINESS AND EARLINESS PROBLEMS

ABSTRACT

We address the problem of determining schedules for static, single-machine scheduling problems where the objective is to minimize the sum of weighted tardiness and weighted earliness. We develop optimal and heuristic procedures for the special case of weights that are proportional to the processing times of the respective jobs. The optimal procedure uses dominance properties to reduce the number of sequences that must be considered, and some of the heuristics use these properties as a basis for constructing good initial sequences. A pairwise interchange procedure is used to improve the heuristic solutions. An experimental study shows that the heuristic procedures perform very well.

Keywords: Scheduling; Integer Programming; Heuristics

ALGORITHMS FOR A CLASS OF SINGLE-MACHINE WEIGHTED TARDINESS AND EARLINESS PROBLEMS

1. INTRODUCTION

Recent research on scheduling problems has been concerned increasingly with objectives related to profitability measures. Two obvious effects of scheduling on profitability are penalties for tardy delivery (or expedited shipment to avoid tardiness) and associated loss of goodwill, and the cost of inventories due to earliness when the finished product cannot be shipped before its due date. The "no early shipment" rule is quite common now with the adoption of just-in-time inventory policies by many firms.

We address the problem of determining schedules for static, single-machine scheduling problems where the objective is to minimize the sum of weighted tardiness and weighted earliness. This objective permits us to reflect the total "lost profit" for each job in a simple and practical way. In the process of developing optimal procedures for the general weighted tardiness and earliness problem, we discovered a subclass of problems for which a simple sorting procedure can provide strong precedence rules, and in some cases, the optimal sequence. The optimal timing of jobs can be found with little effort using a highly structured dynamic program. Since the sorting procedure can provide arbitrarily bad results in the worst case, we develop an optimal algorithm for the subclass of problems and empirically evaluate the sorting procedure in this context.

Relatively little research has been done on scheduling problems involving both tardiness and earliness because the inclusion of earliness in the objective makes it a non-regular measure of performance. Sidney (1977) develops an optimal algorithm for the problem of minimizing the maximum of the costs of earliness and those of tardiness, under the assumption that the costs are non-decreasing functions of earliness and tardiness, respectively. In that paper, earliness is defined as the difference between the target starting time and the actual starting time of a job. A more efficient algorithm for the same problem is given by Lakshminarayan et al. (1978).

Townsend (1978) addresses the problem of minimizing the total penalty, where the penalty for each job is a quadratic function of the completion time. He develops a branch and bound procedure to solve the problem. Gupta and Sen (1983) use the objective of minimizing a quadratic function of job lateness, and present both and optimal branch and bound procedure and a heuristic for this problem. They limit consideration to schedules with no idle time.

A special case of the weighted tardiness and earliness problem in which all weights are equal and the jobs have one common due date, has been discussed in Kanet (1981), Sundararaghavan and Ahmed (1984), and Bagchi et al. (1986). A problem similar to ours is considered by Abdul-Razaq and Potts (1988). They developed a branch and bound algorithm for the weighted tardiness and earliness problem, but do not allow idle time. Since earliness is a non-regular measure of performance, it is possible that delay schedules are optimal, and that the optimal sequence of jobs differs between delay and non-delay schedules. Therefore, decisions about completion times of jobs may be as important as decisions about the sequence.

Fry *et al.* (1987a), Fry *et al.* (1987b), Garey *et al.* (1988), and Kim and Yano (1986) give algorithms for the optimal timing of a given sequence. Fry *et al.* (1987a) consider a problem in which all earliness weights are equal, and all tardiness weights are equal, but the earliness weight differs from the tardiness weight. They give heuristic algorithms in which optimal timing for a sequence is determined through a linear program. Flow times of jobs are considered in addition to earliness and tardiness in Fry *et al.* (1987b). A branch-and-bound procedure is developed and compared with a solution procedure using a mixed integer programming formulation.

Kim and Yano (1986) consider problems in which weights of tardiness and earliness of all jobs are equal. They use a branch and bound algorithm to determine the sequence, and a specially designed algorithm for optimal timing which takes advantage of the problem structure Their optimal timing algorithm runs in O(n^2) time for problems with equal weights, where n is the number of jobs. Using an efficient data structure, Garey *et al.* (1988) achieve $O(n \log n)$ time for both problems with equal weights and different weights. Note that the time complexity of an optimal timing algorithm is important in solving the problem since it may have to be executed many times to find an optimal sequence. They also give an NP-completeness proof for the problem with equal weights for earliness and tardiness of all jobs.

Taking a somewhat different approach, Panwalkar et al. (1982) consider a problem in which due dates are decision variables. They develop a polynomial-time algorithm to find both the sequence and the due dates to minimize costs arising from the selected due dates, earliness and tardiness. Futher details on the above papers can be found in an excellent review by Baker and Scudder (1989).

In the next section we briefly describe a procedure to find the optimal timing of jobs when the sequence is given. We then present a simple sorting procedure and derive conditions in which it is optimal. In the subsequent section, we describe a subclass of problems for which the sorting procedure provides strong precedence information, and we incorporate these ideas into an optimal partial enumeration algorithm. Finally, we present an experimental study in which we evaluate several heuristic procedures and report on the efficiency of the partial enumeration algorithm.

2. DYNAMIC PROGRAMMING PROCEDURE FOR OPTIMAL TIMING

In this section, we assume that the sequence of jobs is given and that the decisions involve only their timing. Let [i] represent the job in the ith position in the sequence, and

$D_{[i]}$	=	due date of job [i],
$p_{[i]}$	=	processing time of job [i],
$\varepsilon_{[i]}$	=	earliness penalty (per unit time) for job [i],
$\tau_{[i]}$	=	tardiness penalty (per unit time) for job [i].

We assume that $\tau_{[i]} \geq \varepsilon_{[i]} \geq 0$, since tardiness normally incurs a larger penalty than does earliness.

The problem can be formulated as a dynamic program with the following recursion formulas:

$$f_n(s) = \min_{s \geq 0} \{ \varepsilon_n (D_n - p_n - s)^+ + \tau_n (s - D_n + p_n)^+ + \min_{t \geq s+p_n} f_{n+1}(t) \}$$

$$f_{N+1}(s) = 0$$

where n denotes the position of the job being scheduled, and s is the (selected) starting time of the job in position n.

Since the penalty function for each job is a piecewise linear convex function of the starting time of the job (also of its completion time), the total penalty function for any subsequence of jobs having no inserted idle time is a piecewise linear, convex function of the starting time of the first job in the subsequence. Consequently, the dynamic program is easy to solve. At each stage, one either schedules the nth job to start at $s = D_n - p_n$ if this would not interfere with jobs n+1,...,N, or schedules it immediately preceding job n+1 and finds the optimal starting time for the subsequence of jobs starting with n and ending with the first idle period. Finding the optimal starting time for this subsequence may result in a concatenation with another subsequence (i.e., an idle period may be eliminated), and the optimal starting time for the larger subsequence would then need to be found.

Computations are further simplified by noting that (i) the slope of the total penalty function can change only when a job changes from being early to being tardy, (ii) the minimum of the function must occur at one of these extreme points, and (iii) the function is convex, so one only needs to consider extreme points until the function starts to increase. It is worth pointing out that the optimal solution (at any stage) may not be unique, but it is advantageous to choose the optimal solution with the latest starting time for the subsequence because this provides the most flexibility to the jobs not yet scheduled. See Kim and Yano (1986) for additional details.

This procedure allows idle time when it is optimal to do so. In the next section, we develop conditions in which a simple sorting procedure produces optimal sequences.

3. A SIMPLE SORTING PROCEDURE

In this section, we show that under certain conditions on the weights and processing times, it is optimal to sequence the jobs in non-decreasing order of $D_i - p_i$ values. Let $s_i = D_i - p_i$ be the target starting time for job i.

Proposition 1. If jobs i and j are adjacent and $s_i \leq s_j$, then it is optimal to sequence job i before job j if

$$\varepsilon_i/\varepsilon_j \leq p_i/p_j, \tag{1}$$

$$p_i/p_j \leq \tau_i/\tau_j, \text{ and} \tag{2}$$

$$\varepsilon_i p_j + \tau_j p_i \leq (\tau_j + \varepsilon_j)(s_j - s_i + p_j) \quad \text{or}$$

$$\varepsilon_i + \tau_i \leq \varepsilon_j + \tau_j \tag{3}$$

Proof. See Appendix A.

The essence of the proof is that if one ordering is preferred to the other for all possible starting times of the earlier of the two jobs, that ordering must occur in the optimal sequence if jobs i and j are adjacent. Also if the three conditions above are satisfied for all pairs of jobs, an optimal sequence can be obtained simply by sorting the jobs in non-decreasing order of the s_i values. If condition (3) is not satisfied for all i and j, sequencing i before j when $s_i \leq s_j$ can have arbitrarily poor performance, as shown in Appendix B We now turn to an description of a subclass of problems for which the conditions in (1) and (2) are automatically satisfied.

4. A CLASS OF PROBLEMS WITH PROPORTIONAL WEIGHTS

Consider a class of problems for which

$$\varepsilon_i = \alpha p_i,$$

and

$$\tau_i = \beta p_i$$

for all i, and for some non-negative real values α and β. This represents a simply-defined but realistic class of problems satisfying conditions (1) and (2). If the value (or cost) added to a job is proportional to its processing time, then the inventory holding cost (earliness penalty) associated with that value added can be viewed as being proportional to the processing time. Sometimes one cannot assess a "value added" because most costs are fixed. In this case, assuming that capacity is constrained (otherwise scheduling would not be a problem), one should consider the opportunity cost associated with the use of the equipment as the "value added," and therefore also proportional to the processing time. In a similar fashion, if the profit from a job is proportional to the processing time, then the opportunity cost of delayed revenue due to delayed shipment (tardiness penalty) should be proportional to the job's processing time. It is also possible that any additional tardiness cost due to loss of goodwill might be proportional to the processing time. Thus, while the conditions may appear to be restrictive, they are sometimes realistic.

Arkin and Roundy (1988a,b) study a special case of this class of problems in which $\alpha = 0$ and $\beta = 1$. They prove that this problem is NP-complete and present an optimal pseudo-polynomial time algorithm for which the running time is quadratic in the number of jobs and linear in the sum of processing times of the jobs. They also present a heuristic, which they call the "earliest-gamma-date" rule, in which jobs are sequenced in increasing order of $\gamma_i = \theta\, s_i + (1-\theta)\, D_i$, where $0 < \theta < 1$. They show that the deviation from the optimal objective value for this heuristic cannot exceed $0.5\,[\theta \vee (1-\theta)]\Sigma_i\, p_i^2$. Consequently, among this class of policies, the best choice of θ is 0.5.

Let us examine in what circumstances the class of problems with $\alpha, \beta > 0$ satisfies condition (3). For this class of problems, condition (3) reduces to

$$p_i - p_j \leq s_j - s_i. \tag{3a}$$

If condition (3a) is satisfied, job i should precede job j. If condition (3a) is not satisfied and $s_i > s_j$, then we must apply Proposition 1 with the subscripts reversed, from which we find that job j should precede job i. In situations where $s_i \leq s_j$ and (3a) does not hold, the optimal resolution of the conflict between jobs i and j depends heavily upon the various penalties. Moreover, in these cases the jobs overlap when scheduled at their respective target starting times, so it is clear that there should be no idle time between the jobs if they are adjacent. We show that if such a conflict occurs and jobs i and j are adjacent, the optimal sequence depends upon *when* the jobs are scheduled (see Appendix C). For this case, i should precede j in the optimal sequence if

1. the selected starting time for the earlier job $\leq s_j - p_i$, or
2. the selected starting time for the earlier job $\geq s_i$.

Simply stated, these conditions say that either both jobs are early or on time, or both jobs are on time or tardy. The two instants of time given above are the breakeven points between i preceding j and j preceding i. If the selected starting time for the earlier job is between $s_j - p_i$ and s_i, job j should precede job i.

It is now evident that for each possible pair of adjacent jobs, we can either specify the optimal ordering *regardless of their timing* if certain conditions hold, or *depending upon their timing* if these conditions do not hold. This information could be used to eliminate many alternative sequences.

We were optimistic that in realistic problems, it would be possible to obtain partial orderings that would permit elimination of most sequences from consideration. With this in mind, we propose a branch-and-bound algorithm in which branching involves assigning available jobs to the last available position in the sequence. (Thus, the procedure constructs the sequence backward.)

The proposed optimal algorithm is a variation of the branch-and-bound algorithm presented in Kim and Yano (1986) for a problem in which all tardiness and earliness weights are equal. Solutions from the heuristic algorithms were used as upper bounds on the optimal solution. The branch and bound tree is constructed by assigning jobs to positions in the sequence starting at the end of the sequence and moving backward. Thus, each node in the branch-and-bound tree is associated with a partial sequence of the last several jobs in the sequence. When we branch from a node, the dominance rules for adjacent jobs are used to avoid considering dominated sequences.

At each node of the branch-and-bound tree, we compute a lower bound on the solution that can be obtained from the partial sequence corresponding to the node. This lower bound is the sum of a lower bound for the jobs *in* the partial sequence and a lower bound for the jobs *not in* the partial sequence. The first lower bound can be obtained by applying an optimal timing algorithm with a constraint that the first job in the partial sequence cannot be started until the earliest possible completion time of the jobs not in the partial sequence. The optimal timing algorithm is a variation of the procedure in Kim and Yano (1986) for the case of equal weights and requires $O(n^2 \log n)$ time in the worst case. The second lower bound is obtained using a property presented in Kim and Yano (1986). Conceptually, this lower bound is simply the sum of unavoidable earliness and tardiness that occurs because two or more jobs would conflict if they were scheduled with $C_i = D_i$ for all i.

Since the problem is a generalization of a problem with tardiness penalties proportional to the processing times and earliness penalties equal to zero, which has been shown to be NP-complete (Arkin and Roundy 1988a), this problem is also. We therefore view the branch and bound algorithm prinicipally as a means to evaluate heuristic solution procedures. In the next section we propose and and test four heuristic procedures and an improvement routine.

5. HEURISTIC PROCEDURES

We compared five heuristic algorithms with the optimal algorithm. Four of the heuristics use sorting procedures to find a sequence. In each case, the final schedule (sequence and timing) is obtained by applying an optimal timing algorithm for each of the sequences. The sorting procedures include EDD (earliest due date) rule, MDD (modified due date) rule, EST (earliest starting time) rule in which jobs are sorted in increasing order of the s_is, and a rule in which the dominance rules are used to establish a sequence. In the last rule, all pairs of two jobs are compared and their priorities (denoted by k_i, i=1,...,n) are computed as follows. Starting with k_i=0 for i=1,...,n, we update the k_is by adding 1 to k_i if job i should precede job j, and subtracting 1 from k_j if job j should succeed job i, where each of these precedence relationships is determined by assuming that the two jobs are adjacent and that their starting times are not restricted. By both adding and subtracting points (rather than simply adding a point for the earlier job in each pair), this procedure tends to provide better discrimination among jobs and to break ties that would occur under simpler rules. After comparing all pairs of jobs, a sequence is obtained by sorting the jobs in decreasing order of the k_i, values. Ties are broken by the EST rule. This rule will be denoted by PREC (short for "precedence").

After applying these four rules on a problem, we start with the best among the four heuristic schedules and apply pairwise interchanges as follows. For k = n-1,...,1, we consider pairwise interchanges of the job originally in position k with the subsequent job in the sequence. If an interchange of the job in position k with the job in position k+1 results in an improvement, the sequence is modified accordingly. An interchange of the job now in position k+1 with the job in position k+2 is considered, and the sequence is modified if appropriate. This process is repeated as long as interchanges result in improvements. This procedure essentially considers moving the job originally in position k to later positions in the sequence by looking forward one position at a time. This is the fifth heuristic tested in this research and is called INT (short for "interchange").

For our experiment, 100 problems were generated randomly with the method used in Potts and Van Wassenhove (1982). In this method, processing times of jobs are randomly generated using a selected distribution, then due dates are generated using the sum of the processing times of all jobs and two parameters, the tardiness factor (T) and relative range (R) of due dates. Following their method, the due dates are generated from a uniform distribution $[\,P(1 - T - R/2), P(1 - T + R/2)\,]$, where P is the sum of the processing times of the jobs. In our test problems, the number of jobs ranged from 7 to 40, the tardiness factor ranged from 0.1 to 0.5, and the relative range of due dates ranged from 0.6 to 1.8. Since only the ratio of the earliness factor to the tardiness factor is of consequence when there are proportional weights, α was set to 1 for all problems and β ranged from 3 to 9.

The algorithms were coded in FORTRAN and run on a personal computer (Zenith Z386). To prevent excessive computation time, the branch-and-bound algorithm was stopped after 3600 seconds (one hour) of CPU time for each problem. In addition, computation was stopped when the memory requirements for a problem exceeded the capacity of the computer memory (640KB of accessible RAM).

The results of the computational experiments are given in Table 1. The table shows the number of jobs, the tardiness factor (T), the relative range (R) of due dates, solution values, and CPU times for each problem. The CPU time given for the heuristic algorithms is the total CPU time required for all five algorithms.

TABLE 1

As shown in the table, a half of the 20-job problems and most of the smaller problems were solved optimally by the branch-and-bound algorithm. Considering the combinatorial aspect of the algorithm, this is a very encouraging result. During testing of the branch-and-bound algorithm, another set of tests was done on small problems. (The results are not shown here.) These tests were designed to study the effects of the dominance rules and the lower bounding methods. The dominance rule in Proposition 1 provided a good pruning tool and eliminated many dominated partial sequences. However, the bounding scheme used for the jobs not in a partial sequence did not perform as well as the dominance rule principally because the lower bound for jobs not in the partial sequence is quite loose. Although the bounds reduced the number of nodes generated (and therefore, the memory size needed for a problem), the computation time was not reduced significantly because of the time required to compute the bound. Further research is needed to develop an improved bounding scheme for this problem.

Compared with the solutions from the branch-and-bound algorithm, the solutions from the heuristic INT were very good. INT found good solutions in a fraction of the CPU time needed for the branch-and-bound algorithm. Only one of the solutions out of 100 was proven to be non-optimal. Even in that problem (Problem 49), the difference between the heuristic and optimal solutions was less than 0.1%. Indeed, we actually had to try to find a problem for which INT produced a solution that was inferior to the incumbent branch and bound solution. It is also important to point out that it is well known that in many branch-and-bound algorithms, the optimal solution is found quickly and the remainder of the CPU time is spent verifying optimality. Thus, although the computation was stopped prior to termination in some cases, the final incumbent solution may be optimal or very close to optimal.

Performance of the five heuristics is further analyzed in Tables 2 and 3. Table 2 shows the average and the worst case performance ratios of the algorithms for different tardiness factors, while Table 3 shows the same for different relative ranges of due dates. The performance ratio in the tables is the ratio of the deviation of the heuristic objective value from optimality to the optimal objective value. If an optimal solution was not found at the termination of the algorithm, the current incumbent solution was used instead of an optimal solution when computing this ratio. Therefore, the ratios in the tables are actually optimistic estimates of the exact ratios.

TABLES 2 AND 3

From Table 2, it is apparent that as the tardiness factor becomes large, the performance of EST and PREC improves, while the performance of EDD and MDD is less sensitive to the tardiness factor. PREC performed best when due dates were tight, and EDD was the best when they were not tight. A similar pattern was observed for sensitivity to the relative range of due dates. While the performance of EDD and MDD did not depend heavily on the relative range, EST and PREC performed better when the range was small. This can be partly explained as follows. When the due dates of jobs are dispersed, there would be little conflict between jobs even if they were scheduled to be completed at their due dates. Therefore, the solution from EDD sequence may be close to an optimal solution. On the other hand, when the due dates are not dispersed widely, they do not play as much of a role in determining the sequence; the relative magnitudes of the penalties are more important. These results corroborate findings on the tardiness minimization problem. In the tardiness problem, when two jobs are both late, it is better to sequence the two jobs in SPT (shortest processing time) order rather than EDD order.

All of the above results demonstrate the importance of including a pairwise interchange method in the heuristic algorithm. It can enhance the performance of a heuristic without much additional computation time. In our algorithm, the interchange routine runs in $O(n^2)$ time.

6. CONCLUSIONS

We have developed optimal and heuristic procedures for the problem of minimizing the sum of weighted tardiness and weighted earliness on a single machine when the weights are proportional to the processing times of the jobs. Dominance criteria are derived which eliminate many possible sequences from consideration in the optimal branch-and-bound procedure, and provide a basis for determining an initial sequence and evaluating potential improvements in the heuristic procedures. Although the problem is known to be NP-complete, the dominance criteria permitted us to optimally solve many problems with 15 jobs or more. Further research to develop a tighter lower bound may permit much larger problems to be solved optimally.

A composite heuristic which combines several sorting routines and a simple pairwise interchange procedure performed extremely well in computational tests. In only one out of 100 problems did the optimal algorithm (with a generous CPU time trap) find a better solution than the heuristic procedure. Additional research is needed to develop dominance criteria and heuristic procedures for the more general weighted tardiness and earliness problem.

Table 1. Performance of the algorithms on 100 test problems

				Heuristic algorithms						Optimal algorithm	
Problem	n	T	R	EDD	EST	MDD	PREC	INT	CPU	B&B	CPU
1	7	.5	1.0	1373	1325	1373	1325	1325	.05	1325	.93
2	7	.4	1.0	719	719	719	719	719	.05	719	.60
3	8	.5	1.0	23884	23664	23884	23664	23664	.05	23664	3.07
4	8	.4	1.2	1217	1217	1217	1217	1217	.05	1217	.71
5	8	.3	1.2	6564	6528	6564	6528	6528	.05	6528	1.37
6	10	.5	1.0	12956	12920	13166	12920	12920	.06	12920	8.63
7	10	.4	1.2	1955	1955	1955	1955	1955	.06	1955	4.83
8	10	.3	1.4	11563	11323	12331	11323	11323	.11	11323	2.14
9	10	.2	1.6	6824	6508	6824	6880	6508	.05	6508	11.09
10	10	.1	1.8	2952	4374	2952	3096	2760	.05	2760	12.96
11	10	.5	.9	6450	6450	6450	6450	6450	.06	6450	2.14
12	10	.5	.8	3198	3174	3198	3174	3174	.11	3174	2.14
13	10	.5	.7	5787	5667	7101	5667	5667	.11	5667	2.96
14	10	.4	1.0	1131	1323	1131	1323	1131	.06	1131	1.21
15	10	.4	.8	8021	8007	8861	7895	7895	.11	7895	9.01
16	12	.4	.6	8869	7602	8869	8435	7602	.05	7602	33.50
17	12	.3	1.2	5918	5724	5918	5852	5598	.11	5598	4.23
18	12	.3	1.0	7401	8874	7401	8874	7401	.05	7401	5.55
19	12	.3	.8	11745	11775	11745	11927	11745	.11	11745	106.89
20	12	.2	1.4	2196	2196	2196	2196	2196	.06	2196	2.26
21	14	.2	1.2	11536	12157	11536	11536	11536	.05	11536	62.83
22	14	.2	1.0	6298	10636	6298	9376	6298	.11	6298	92.60
23	14	.1	1.5	10174	13484	10174	11014	10174	.11	10174	76.46
24	14	.1	1.2	13493	13209	13493	13209	12509	.17	12509	666.91
25	14	.1	.9	40343	40343	40343	40343	40343	.11	40343	459.39
26	15	.5	1.0	3674	3590	3872	3590	3590	.11	3590	41.31
27	15	.4	1.2	5188	6065	5272	5314	5188	.11	5188	40.09
28	15	.3	1.4	13720	13991	13720	13991	13656	.16	13656	97.71
29	15	.2	1.6	50102	50102	50858	50102	50102	.11	50102	111.01
30	15	.1	1.8	13051	14351	13051	14351	13051	.05	13051	47.68
31	15	.5	.9	4551	4491	4731	4359	4359	.22	4359	47.40
32	15	.5	.8	4026	3978	4080	3858	3858	.17	3858	5.55
33	15	.5	.7	9095	9305	9491	9305	9005	.22	9005	135.45
34	15	.4	1.0	2519	2827	2519	2827	2519	.11	2519	10.93
35	15	.4	.8	11759	11759	12683	11759	11759	.16	11759	28.29
36	15	.4	.6	17575	17106	17575	17106	17106	.22	17106	863.49
37	15	.3	1.2	13811	14291	13811	14291	13811	.16	13811	567.49
38	15	.3	1.0	12224	212176	13576	12224	12176	.11	12176	28.73
39	15	.3	.8	9922	10339	9922	10339	9922	.17	9922	300.66
40	15	.2	1.4	16456	14034	18220	14034	13936	.11	13936	34.99
41	15	.2	1.2	6890	7279	6890	7279	6174	.11	6174	204.43
42	15	.2	1.0	28107	28107	28107	28107	28107	.11	28107	2433.80
43	15	.1	1.5	3243	3311	3243	3311	3243	.05	3243	39.76
44	15	.1	1.2	6766	6994	6766	6994	6766	.11	6766	238.00
45	15	.1	.9	43138	43950	43138	41550	41550	.16	*41550	3600.
46	16	.5	.9	16371	16371	16857	16371	1637	.11	16371	52.35
47	16	.5	.8	9034	9034	9034	9034	9034	.17	9034	60.86
48	16	.5	.7	9556	9556	9610	9556	9556	.11	9556	352.84
49	16	.4	1.0	15466	15301	15466	15301	15301	.17	15298	383.38
50	16	.4	.8	6870	6864	6870	6864	6864	.17	6864	260.90
51	18	.4	.6	24895	24993	25161	24769	24769	.33	*24769	3600.

Table 1.(continued)

				Heuristic algorithms						Optimal algorithm	
Problem	n	T	R	EDD	EST	MDD	PREC	INT	CPU	B&B	CPU
52	18	.3	1.2	4833	5093	4833	5093	4333	.17	4333	428.37
53	18	.3	1.0	6072	6274	6072	6274	6072	.11	6072	1014.64
54	18	.3	.8	10795	10920	10795	10920	10795	.17	*10795	2649.06
55	18	.2	1.4	16685	16873	16685	16685	16685	.22	*16685	2411.23
56	20	.5	1.0	7236	7236	7278	7236	7236	.16	7236	3315.96
57	20	.4	1.2	12507	12507	15048	12507	12507	.16	12507	31.03
58	20	.3	1.4	30705	31559	30705	31559	30705	.22	*30705	2440.13
59	20	.4	1.6	5622	9425	5622	9425	5622	.17	5622	195.43
60	20	.1	1.8	6048	9898	6048	9898	6048	.16	6048	489.93
61	20	.5	.9	4613	4291	4613	4291	4291	.17	4291	328.07
62	20	.5	.8	23955	23955	24945	23955	23955	.27	23955	2952.13
63	20	.5	.7	8406	8754	9264	8628	8406	.27	8406	1994.84
64	20	.4	1.0	8734	9838	8734	9838	98734	.22	8734	3227.37
65	20	.4	.8	19604	19758	20976	19478	19373	.39	*64123	2733.92
67	20	.3	1.2	35551	33927	37319	33927	33927	.22	33927	206.90
68	20	.3	1.0	5960	6043	5960	5960	5960	.22	* 5960	2837.67
69	20	.3	.8	18755	18755	18755	18755	18755	.22	*18755	2711.12
70	20	.2	1.4	12137	12767	12137	12767	12137	.16	12137	2136.98
71	20	.2	1.2	15870	15777	15870	14064	14064	.22	*14064	2923.91
72	20	.2	1.0	11121	12816	11121	12816	11121	.16	*11121	2241.73
73	20	.1	1.5	14264	17924	14264	17924	14264	.17	*14264	2543.49
74	20	.1	1.2	17192	25478	17912	25478	17912	.22	*17912	2575.13
75	20	.1	.9	52419	56769	52419	54099	52419	.27	*52419	2374.35
76	20	.2	1.2	55709	55804	57842	55804	55709	.22	55709	344.28
77	20	.2	1.0	18946	19945	18946	19549	18946	.22	*18946	2399.81
78	20	.1	1.5	14740	20173	14740	19259	14740	.17	14740	987.29
79	20	.1	1.2	13911	16241	13911	16241	13911	.22	*13911	2356.19
80	20	.1	.9	24121	29887	24121	29887	24121	.22	*24121	2136.66
81	25	.5	1.0	3574	3937	3574	3886	3574	.27	* 3574	3600.
82	25	.4	1.2	17614	17590	17614	17590	17320	.38	*17320	2256.83
83	25	.3	1.4	15704	17047	15704	16963	15688	.33	*15688	2691.84
84	25	.2	1.6	11704	13912	11704	13912	11704	.22	*11704	3600.
85	25	.1	1.8	6448	6912	6448	6884	6448	.22	* 6448	3600.
86	30	.5	1.0	35685	35685	37107	35685	35685	.50	*35685	2565.02
87	30	.4	1.2	15112	14978	15492	14908	14433	.39	14433	1523.47
88	30	.3	1.4	15828	16644	15828	16644	15828	.33	*15828	3600.
89	30	.2	1.6	23063	27130	30758	27130	23063	.33	*23063	3600.
90	30	.1	1.8	19411	19801	19411	19801	19411	.33	*19411	3600.
91	35	.5	1.0	9806	9770	9806	9770	9770	.44	* 9770	3600.
92	35	.4	1.2	10366	10996	10366	10996	10366	.60	*10366	3600.
93	35	.3	1.4	58104	58960	58296	58960	58104	.60	*58104	2530.14
94	35	.2	1.6	18106	21586	18229	21586	18106	.44	*18106	3600.
95	35	.1	1.8	57313	56921	57313	56921	56837	.49	*56837	3600.
96	40	.5	1.0	21753	21747	22899	21747	21747	.93	*21747	2896.00
97	40	.4	1.2	54287	53703	55085	53584	53496	1.05	*53496	3034.97
98	40	.3	1.4	33161	33621	35153	33349	33161	.66	*33161	3203.09
99	40	.2	1.6	50796	53699	52749	53699	50796	.60	*50796	3137.67
100	40	.1	1.8	24751	26780	24751	26780	24751	.66	*24751	3600.

* This is the incumbent solution at the time when the algorithm was stopped.

Table 2. Performance ratios of the algorithms for each tardiness factor

	# of	Average Performance Ratios					Worst Case Performance Ratios				
T	probs	EDD	EST	MDD	PREC	INT	EDD	EST	MDD	PREC	INT
.1	19	.0103	.1856	.0103	.1593	.0	.0786	.6357	.0787	.6366	.0
.2	19	.0249	.1374	.0544	.1190	.0	.1808	.6888	.3337	.6765	.0
.3	20	.0128	.0377	.0276	.0383	.0	.1154	.1990	.1154	.1990	.0
.4	21	.0151	.0357	.0415	.0319	.00001	.1667	.1698	.2032	.1698	.00020
.5	21	.0132	.0113	.0456	.0070	.0	.0750	.1016	.2530	.0873	.0
Total	100	.0152	.0788	.0361	.0699	.00000	.1808	.6888	.3337	.6765	.00020

Table 3. Performance ratios of the algorithms for each relative range

	# of	Average Performance Ratios					Worst Case Performance Ratios				
R	probs	EDD	EST	MDD	PREC	INT	EDD	EST	MDD	PREC	INT
.6	4	.0499	.0023	.0554	.0274	.0	.1667	.0090	.1667	.1096	.0
.7	4	.0078	.0187	.1037	.0149	.0	.0212	.0414	.2530	.0333	.0
.8	12	.0067	.0101	.0326	.0062	.0	.0436	.0420	.1224	.0430	.0
.9	8	.0197	.0513	.0285	.0339	.0	.0750	.0239	.0863	.2390	.0
1.0	22	.0041	.0766	.0169	.0643	.00001	.0362	.6888	.1150	.4887	.00020
1.2	21	.0299	.0741	.0473	.0596	.0	.1284	.4224	.2032	.4224	.0
1.4	11	.0189	.0263	.0423	.0241	.0	.1808	.0866	.3074	.0813	.0
1.5	4	.0	.2426	.0	.1667	.0	.0	.3679	.0	.3065	.0
1.6	7	.0069	.1844	.0732	.1926	.0	.0486	.6765	.3337	.6765	.0
1.8	7	.0111	.2138	.0111	.1470	.0	.0696	.6366	.0696	.6366	.0
Total	100	.0152	.0788	.0361	.0699	.00000	.1808	.6888	.3337	.6765	.00020

REFERENCES

Abdul-Razaq, T.S. and C.N. Potts (1988), "Dynamic Programming State-Space Relaxation for Single Machine Scheduling," *Journal of the Operational Research Society* 39(2), 141-152.

Arkin, E.M. and R.O. Roundy (1988a), "A Pseudo-Polynomial Time Algorithm for Weighted-Tardiness Scheduling with Proportional Weights," Technical Report No. 812, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY.

Arkin, E.M. and R.O. Roundy (1988b), "Weighted-Tardiness Scheduling on Parallel Machines with Proportional Weights," Technical Report No. 768, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY.

Bagchi, V., R.S. Sullivan, and Y.L. Chang (1986), "Minimizing Mean Absolute Deviation of Completion Times about a Common Due Date," *Naval Research Logistics* 22(3), 585-592.

Baker, K.R. and G.D. Scudder (1988), "Sequencing with Earliness and Tardiness Penalties: A Review," Working Paper No. 226, The Amos Tuck School of Business Administration, Dartmouth College, Hanover, NH.

Fry, T. D., R. D. Armstrong, and J. H. Blackstone (1987a), "Minimizing Weighted Absolute Deviation in Single Machine Scheduling", *IIE. Transactions* 19(4), 445-450.

Fry, T. D., G. K. Leong, and T. R. Rakes (1987b), "Single Machine Scheduling: A Comparison of Two Solution Procedures", *OMEGA* 15(4), 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* 13(2), 330-348.

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

Kanet, J.J. (1981), "Minimizing the Average Deviation of Job Completion Times about a Common Due Date," *Naval Research Logistics Quarterly* 28(4), 643-651.

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

Lakshminarayan, S., R. Lakshmanan, R.L. Papineau, and R. Rochette (1978), *Operations Research* 26(4), 1079-1082.

Panwalkar, S.S., M.L. Smith, and A. Seidmann (1982), "Common Due Date Assignment to Minimize Total Penalty for the One Machine Scheduling Problem," *Operations Research* 30(2), 391-399.

Potts, C. N. and L. N. Van Wassenhove (1982), "A Decomposition Algorithm for the Single Machine Total Tardiness Problem", *Operations Research Letters* 1(5), 177-181.

Sundararaghavan, P.S. and M.U. Ahmed (1984), "Minimizing the Sum of Absolute Lateness in Single-Machine and Multimachine Scheduling," *Naval Research Logistics Quarterly* 31(2), 325-333.

Townsend, W. (1978), "The Single Machine Problem with Quadratic Penalty Function of Completion Times: A Branch and Bound Solution," *Management Science* 24(5), 530-534.

APPENDIX A
PROOF OF PROPOSITION 1

We initially consider the case of no idle time between jobs and then extend the results to the case of permitted idle time. Recall that for each i, $\varepsilon_i E_i + \tau_i T_i$ is a piecewise linear convex function of the completion time (and therefore also the starting time) of job i. Consider two jobs, i and j, processed sequentially. For each of the two orderings, total weighted earliness and tardiness (which refer to as total cost) is a piecewise linear convex function of the starting time of the earlier job.

Observe that for sufficiently early starting times, both jobs will be early, and the total cost declines (for a while) at a rate of $\varepsilon_i + \varepsilon_j$ as the starting time is increased. Similarly, for sufficiently late starting times, both jobs will be tardy and the total cost increases by $\tau_i + \tau_j$ as the starting time is increased. Thus, for sufficiently small and large starting times, the total cost has the same slope, regardless of the ordering. This fact, in conjunction with the piecewise linear convex nature of the cost functions makes it sufficient to compare the two cost functions at all extreme points to establish that one dominates the other. This is the approach used in our proof.

There are only four starting times at which extreme points can (but do not necessarily) occur: (1) s_i; (2) s_j; (3) $s_i - p_j$; (4) $s_j - p_i$. At time s_i, if job i is scheduled first, it changes from being early to being tardy. Similarly, at s_j, job j changes from being early to being tardy if it is scheduled first. At a starting time of $s_i - p_j$, job i changes from being early to being tardy if job j is scheduled first. Likewise, at a starting time of $s_j - p_i$, job j changes from being early to being tardy if job i is scheduled first.

There are three possible relationships among the four possible extreme points in (1) through (4) above:

Case 1) $s_i - p_j < s_i \leq s_j - p_i < s_j$,

Case 2) $s_i - p_j \leq s_j - p_i \leq s_i \leq s_j$, and

Case 3) $s_j - p_i \leq s_i - p_j < s_i \leq s_j$.

The values of the cost function at each possible extreme point can be obtained easily for each of the two possible job sequences. These values are shown in Tables A1 through A3 for the three cases listed above.

Table A1. Total cost at possible extreme points
Case 1 ($s_i - p_j < s_i \leq s_j - p_i < s_j$)

Starting time	Sequence	
	i before j	j before i
$s_i - p_j$	$\varepsilon_i p_j + \varepsilon_j(s_j - s_i - p_i + p_j)$	$\varepsilon_j(s_j - s_i + p_j)$
s_i	$\varepsilon_j(s_j - s_i + p_i)$	$\tau_i p_j + \varepsilon_j(s_j - s_i)$
$s_j - p_i$	$\tau_i(s_j - p_i - s_i)$	$\tau_i(s_j - p_i + p_j - s_i) + \varepsilon_j p_i$
s_j	$\tau_i(s_j - s_i) + \tau_j p_i$	$\tau_i(p_j + s_j - s_i)$

Table A2. Total cost at possible extreme points
Case 2 ($s_i - p_j \leq s_j - p_i < s_i \leq s_j$)

Starting time	Sequence	
	i before j	j before i
$s_i - p_j$	$\varepsilon_i p_j + \varepsilon_j(s_j - s_i + p_j - p_i)$	$\varepsilon_j(s_j - s_i + p_j)$
$s_j - p_i$	$\varepsilon_i(s_i - s_j + p_i)$	$\tau_i(s_j - s_i + p_j - p_i) + \varepsilon_j p_i$
s_i	$\tau_j(s_i - s_j + p_i)$	$\tau_i p_j + \varepsilon_j(s_j - s_i)$
s_j	$\tau_i(s_j - s_i) + \varepsilon_j p_i$	$\tau_i(s_j - s_i + p_j)$

Table A3. Total cost at possible extreme points
Case 3 ($s_j - p_i \le s_i - p_j < s_i \le s_j$)

Starting time	Sequence	
	i before j	j before i
$s_j - p_i$	$\varepsilon_i(s_i - s_j + p_i)$	$\varepsilon_i(s_i - s_j - p_j + p_i) + \varepsilon_j p_i$
$s_i - p_j$	$\varepsilon_i p_j + \tau_j(s_i - s_j + p_i - p_j)$	$\varepsilon_j(s_j - s_i + p_j)$
s_i	$\tau_j(s_i - s_j + p_i)$	$\tau_i p_j + \varepsilon_j(s_j - s_i)$
s_j	$\tau_i(s_j - s_i) + \tau_j p_i$	$\tau_i(s_j - s_i + p_j)$

Optimality of the schedule in which i precedes j can be proved by simply comparing the total costs of the two orderings, using Tables A1 through A3.

Case 1)

starting time $s_i - p_j$: i precedes j if $\varepsilon_i p_j - \varepsilon_j p_i \le 0$ or $\varepsilon_i/\varepsilon_j \le p_i/p_j$.

starting time s_i: i precedes j for all $\varepsilon_j \ge 0$, $\tau_i \ge 0$

starting time $s_j - p_i$: i precedes j for all $\tau_i \ge 0$, $\varepsilon_j \ge 0$

starting time s_j: i precedes j if $\tau_j p_i \le \tau_i p_j$ or $p_i/p_j \le \tau_i/\tau_j$

Case 2)

starting time $s_i - p_j$: i precedes j if $\varepsilon_i p_j - \varepsilon_j p_i \le 0$ or $\varepsilon_i/\varepsilon_j \le p_i/p_j$

starting time $s_j - p_i$: i precedes j if $\varepsilon_i(s_i - s_j + p_i) \le \tau_i(s_j - p_i + p_j - s_i) + \varepsilon_j p_i$. Rewriting the condition, we have $(\tau_i + \varepsilon_i)(s_i - s_j + p_i) \le \tau_i p_j + \varepsilon_j p_i$, where $0 \le s_i - s_j + p_i \le p_j$ by definition. Thus, $(\tau_i + \varepsilon_i)(s_i - s_j + p_i) \le (\tau_i + \varepsilon_i)p_j$, and $(\tau_i + \varepsilon_i)p_j \le \tau_i p_j + \varepsilon_j p_i$ if $\varepsilon_i p_j \le \varepsilon_j p_i$ or $\varepsilon_i/\varepsilon_j \le p_i/p_j$.

starting time s_i: i precedes j if $\tau_j(s_i - s_j + p_i) \le \tau_i p_j + \varepsilon_j(s_j - s_i)$ or $\tau_j p_i \le \tau_i p_j + (\tau_j + \varepsilon_j)(s_j - s_i)$. Since $s_j - s_i \ge 0$, the second term on the right hand side is non-negative. The only condition is $p_i/p_j \le \tau_i/\tau_j$.

starting time s_j: i precedes j if $\tau_j p_i \le \tau_i p_j$ or $p_i/p_j \le \tau_i/\tau_j$.

Case 3)

starting time $s_j - p_i$: i precedes j if $\varepsilon_j p_i - \varepsilon_i p_j \geq 0$ or $\varepsilon_i/\varepsilon_j \leq p_i/p_j$

starting time $s_i - p_j$: i precedes j if $\varepsilon_i p_j + \tau_j(s_i - s_j + p_i - p_j) \leq \varepsilon_j(s_j - s_i + p_j)$ or $\varepsilon_i p_j + \tau_j p_i \leq (\varepsilon_j + \tau_j)(s_j - s_i + p_j)$.

If $\tau_i p_j \geq \tau_j p_i$ then the left hand side of the inequality is less than or equal to $(\varepsilon_i + \tau_i)p_j$. Since $s_j - s_i \geq 0$, in this situation a simpler sufficient condition is $\varepsilon_i + \tau_i \leq \varepsilon_j + \tau_j$.

starting time s_i: i precedes j if $\tau_j(s_i + p_i - s_j) \leq \tau_i p_j + \varepsilon_j(s_j - s_i)$ or $\tau_j p_i \leq \tau_i p_j + (\tau_j + \varepsilon_j)(s_j - s_i)$. Since $s_j - s_i \geq 0$, this is satisfied if $p_i/p_j \leq \tau_i/\tau_j$.

starting time s_j: i precedes j if $\tau_j p_i \leq \tau_i p_j$ or $p_i/p_j \leq \tau_i/\tau_j$.

We next sketch a proof to show that the above results are true even when idle time between jobs i and j is permitted. When both jobs are early, the schedule can be improved by permitting idle time between the jobs. However, it is easy to show (proof omitted here) that the schedule with idle time is dominated by another feasible schedule with a later starting time and without idle time between the two jobs. Thus, the analysis for the no idle time case applies here.

In all other cases, either (1) the schedule for neither sequence can be improved by permitting idle time, or (2) the schedule for the sequence with i before j can be improved by permitting idle time, but the schedule for the other sequence cannot. (The proof is tedious but straightforward.) Hence, the differences between the costs reflected in Tables A1 through A3 represent upper bounds on the errors.

This, along with the proof for the case of no idle time between jobs, completes the proof of the proposition.

APPENDIX B

Worst Case Analysis of the Sorting Procedure

In this Appendix, we demonstrate that in the worst case, sorting jobs in non-decreasing order of the s_i values can have arbitrarily poor performance. Consider a situation in which $s_j - p_i \leq s_i - p_j < s_i \leq s_j$, and suppose that the optimal schedule has j preceding i with job j starting in $(s_j - p_i, s_i]$. In this case, both jobs are early and the penalty is

$$\varepsilon_j (s_j - t_j) + \varepsilon_i (s_i - t_i)$$

where t_j is the actual starting time of job j and t_i is the actual starting time of job i.

Since both jobs are early, it is advantageous to have no idle time between the jobs and to delay both jobs at least until one of the two (in this case, job i) is exactly on time ($t_j = s_i - p_j$, $t_i = s_i$). Suppose now that $\tau_i > \varepsilon_i > \varepsilon_j$, so it is undesirable to make job i tardy in order to reduce the earliness of job j. Then, the optimal solution is the one given above, and the associated penalty is

$$\varepsilon_j (s_j + p_j - s_i).$$

Now consider what happens when the sorting procedure results in job i preceding job j, with job i starting after s_i. In this case, both jobs are tardy. As such, it is optimal to have no idle time between the jobs and the penalty is

$$\tau_i (t_i - s_i) + \tau_j (t_i + p_i).$$

Since we cannot determine the value of t_i *a priori*, this penalty can be arbitrarily large (although clearly finite).

APPENDIX C
Analysis of Situations where Job j Should Precede Job i when $s_i < s_j$.

Recall that condition (3) in Appendix A pertains to situations where $s_j - p_i \leq s_i - p_j < s_i \leq s_j$. We have already shown that under the condition of proportional weights, if condition (3) holds and jobs i and j are adjacent, job i should precede job j if the starting time of the earlier is less than or equal to $s_j - p_i$ or greater than or equal to s_i. Let us analyze what happens when the starting time of the earlier job is between the two points.

For any starting time $t \; \varepsilon \; (s_j - p_i, s_i - p_j]$, the cost if job i immediately precedes job j (with no idle time between them) is

$$\varepsilon_i(s_i - t) + \tau_j(t + p_i - s_j)$$

and the cost if job j precedes job i is

$$\varepsilon_j(s_j - t) + \varepsilon_i(s_i - t - p_j).$$

Thus, the sequence with i before j is preferred if

$$\varepsilon_i(s_i - t) + \tau_j(t + p_i - s_j) \leq \varepsilon_j(s_j - t) + \varepsilon_i(s_i - t - p_j).$$

Using the fact that $\varepsilon_i p_j = \alpha p_i p_j = \varepsilon_j p_i$, this simplifies to

$$t \leq s_j - p_i,$$

which is inconsistent with the original assumption about t. Therefore, if jobs i and j are scheduled with one immediately after the other, it is better to put job j first under the conditions given above. Moreover, it is obvious that since both jobs are early when job j is sequenced first, the cost associated with that sequence can be reduced by inserting idle time between the two jobs. In the other sequence, however, one job is early and the other is tardy, Hence, it is optimal to have no idle time between the jobs, and its cost cannot be reduced further. Consequently, the conditions given above reflect a case where job j should always precede job i.

For a starting time $t \varepsilon (s_i - p_j, s_i)$, the cost if job i precedes job j is

$\varepsilon_i(s_i - t) + \tau_j(t + p_i - s_j)$

and the cost if job j precedes job i is

$\varepsilon_j(s_j - t) + \tau_i(t + p_j - s_i)$.

Using arguments similar to those above, it can be shown that the sequence with j before i is always preferred under the stated conditions.