EARLINESS-TARDINESS SCHEDULING PROBLEMS, I: WEIGHTED DEVIATION OF COMPLETION TIMES ABOUT A COMMON DUE DATE

NICHOLAS G. HALL and MARC E. POSNER

The Ohio State University, Columbus, Ohio

(Received March 1989; revisions received September 1989, January 1990; accepted April 1990)

This paper and its companion (Part II) concern the scheduling of jobs with cost penalties for both early and late completion. In Part I, we consider the problem of minimizing the weighted sum of earliness and tardiness of jobs scheduled on a single processor around a common due date, d. We assume that d is not early enough to constrain the scheduling decision. The weight of a job does not depend on whether the job is early or late, but weights may vary between jobs. We prove that the recognition version of this problem is NP-complete in the ordinary sense. We describe optimality conditions, and present a computationally efficient dynamic programming algorithm. When the weights are bounded by a polynomial function of the number of jobs, a fully polynomial approximation scheme is given. We also describe four special cases for which the problem is polynomially solvable. Part II provides similar results for the unweighted version of this problem, where d is arbitrary.

A wide variety of problems conform closely to the scheduling environment we consider here. There are n jobs which, ideally, would all be completed precisely at a known due date. Costs or penalties are incurred by the completion of a job either earlier or later than the due date. Costs of earliness could result, for example, from deterioration or the need for storage and insurance. In addition, finished goods inventory represents an unproductive investment which implicitly incurs an opportunity cost. The costs of lateness, or tardiness, are even more familiar, and include customer dissatisfaction, contract penalties, loss of sale, and potential loss of reputation. In view of the generality of this deterministic scheduling environment, and its compatibility with current developments such as just-in-time (Henderson 1982, Porteus 1985) in inventory management, the recent growth of interest is understandable.

For a comprehensive review of machine scheduling with costs for both early and late completion of jobs, see Baker and Scudder (1990). One of the earliest papers on this topic is by Sidney (1977), who provides an efficient algorithm to minimize the maximum earliness or tardiness penalty on a single machine, where jobs have a restricted variety of due dates. This algorithm is improved by Lakshminarayan et al. (1978). Extensions of this problem to a flow shop scheduling environment are discussed by Achutan, Grabowski and Sidney (1981). Ow and Morton (1989) provide a computational study of several heuristics for minimizing total weighted earliness and tardiness. The origins of a different research direction can be traced to the work of Kanet (1981). He considers the problem of minimizing the total (unweighted) earliness and tardiness around a single common due date. The assumption made is that the due date is "unrestrictively late," that is, not early enough to act as a constraint on the scheduling decision. Under this assumption, Kanet provides an algorithm for finding an optimal solution in polynomial time.

The optimality conditions for Kanet's problem, and the availability of a large number of optimal solutions, are discussed by Hall (1986), and Bagchi, Sullivan and Chang (1986). Emmons (1987) examines secondary criteria as a way of choosing among the optimal solutions. Sundararaghavan and Ahmed (1984) generalize Kanet's problem to a scheduling environment with several identical parallel machines. Bagchi (1989) discusses the bicriterion problem of simultaneously minimizing the mean and variation of flow time. Kubiak, Lou and Sethi (1990) show that the weighted earliness and tardiness problem on unrelated parallel machines can be solved efficiently as a transportation problem. Another generalization of Kanet's problem is studied by Bagchi, Chang and Sullivan (1987), where all jobs have equal earliness and tardiness

Subject classifications: Dynamic programming, deterministic: earliness and lateness costs. Production/scheduling: deterministic sequencing; single machine.

0030-364X/91/3905-0836 $01.25

weights. The authors describe optimality conditions which, in the case where the due date is unrestrictively late, characterize an efficient algorithm.

An alternative performance measure, minimization of mean squared deviation, is considered by Bagchi, Sullivan and Chang (1987). Another variant of Kanet's problem, where the common due date is restrictively early, is considered by Szwarc (1989). Properties of optimal solutions are discussed in that paper, and a computationally efficient branch-and-bound procedure is also described.

Garey, Tarjan and Wilfong (1988) consider the problem of minimizing total (unweighted) earliness and tardiness on a single machine, where due dates may vary between jobs. This important work contains one of the first proofs of NP-completeness for a single machine scheduling problem with a nonregular performance measure (see Garey and Johnson 1979, or Papadimitriou and Steiglitz 1982). Efficient algorithms are provided if all job lengths are equal, or if jobs must be executed in a known sequence. Similar definitions of the boundary between easy and hard problems for the scheduling environment without earliness costs are given by Lenstra, Rinnooy Kan and Brucker (1977), for example.

The problem we study here is a direct generalization of Kanet's problem, where each job j has a weight or value, w_j, which is "symmetric," i.e., independent of whether the job is early or late. The problem is more restricted than the problem considered in Garey, Tarjan and Wilfong because all the due dates here have the same value which is unrestrictively late. This falls in the gap between known polynomially solvable and NP-complete problems, and is one of the few scheduling problems whose complexity remains open.

In this paper, we describe optimality conditions. We prove that the recognition problem corresponding to the minimization of total weighted earliness and tardiness is NP-complete in the ordinary sense. Then, we develop a pseudopolynomial time algorithm, and show that the procedure is computationally efficient. We solve randomly generated scheduling problems with 2,000 jobs in under two minutes of CPU time on a minicomputer. Next, a fully polynomial approximation scheme is developed under general assumptions of the boundedness of the data. Lastly, polynomial time algorithms are developed for four special cases.

1. OPTIMALITY CONDITIONS

The weighted earliness and tardiness problem (**WET**) is defined by: n jobs with known integer weights $w_1, w_2, \ldots, w_n$, processing times $p_1, p_2, \ldots, p_n$, and a common due date d, where d is unrestrictively large. Since there are problems where the optimal solution requires all jobs to complete by d (see Section 5.3), we assume that $d \geq \sum_{j=1}^{n} p_j$. Let C_j denote the completion time of job j in schedule σ for $j = 1, 2, \ldots, n$, and $z(\sigma)$ be the cost of σ. The objective is to minimize the sum of weighted earliness and tardiness of the jobs, that is:

$$\min_{\sigma \in \Pi} z(\sigma) = \sum_{j=1}^{n} w_j | d - C_j |$$

where Π denotes the set of all possible nonpreemptive single machine schedules.

In this section, we describe several optimality conditions for **WET**. Since any schedule with idle time between jobs can have its cost reduced by the removal of gaps, we only consider schedules that process jobs consecutively. Define $E = \{1 \leq j \leq n \mid C_j < d\}$ and $T = \{1 \leq j \leq n \mid C_j > d\}$.

Property 1. *There exists an optimal solution to* **WET**, *where some job completes at d.*

Proof. Assume that we have an optimal schedule where no job finishes at d. Let i be the job that is being processed at d, and $0 < \epsilon < \min\{d - (C_i - p_i), C_i - d\}$. Let the change in cost from starting all jobs ϵ time units later be $\Delta_1 = \epsilon(\sum_{j \in T} w_j - \sum_{j \in E} w_j)$. Similarly, let $\Delta_2 = \epsilon(\sum_{j \in E} w_j - \sum_{j \in T} w_j) = -\Delta_1$, where Δ_2 denotes the change in cost from starting all jobs ϵ time units earlier. Clearly, $\Delta_1 \leq 0$ or $\Delta_2 < 0$. Slide all the jobs later if $\Delta_1 \leq 0$ or earlier if $\Delta_2 < 0$ until some job completes at d. Since the cost does not increase, this schedule is also optimal.

Henceforth, we only consider schedules where some job completes at d. We indicate that job i_k completes at d by placing @ after the job in the schedule, for example, $(i_1, i_2, \ldots, i_{k-1}, i_k@, i_{k+1}, \ldots)$. Define E′ as $E \cup \{j \mid C_j = d\}$.

We assume throughout that the jobs are ordered so that $w_1/p_1 \geq w_2/p_2 \geq \ldots \geq w_n/p_n$. The next result states how to find an optimal solution given that E′ and T are fixed.

Property 2. *There exists an optimal schedule where the jobs in* E′ *are in decreasing index order and the jobs in* T *are in increasing index order.*

Proof. The proof follows from a job interchange argument.

The ordering given by Property 2 is the well known **V-schedule** that is characteristic of many nonregular measure, one-machine scheduling problems (see Eilon and Chowdhury 1977, for instance). Jobs before d have nondecreasing w_j/p_j ratios, whereas jobs after d have nonincreasing w_j/p_j ratios.

In Property 2, we consider the ordering of the jobs within E′ and T. In Properties 3–6, we examine relationships between the two sets. For each of these properties, we define an optimal schedule σ^*, where job s is the first job processed, job t is the job that finishes at d, and job u is the last job processed. This schedule is illustrated in Figure 1. In the proofs, various possible adjustments to σ^* are examined. We let Δ denote the resulting change in cost in each case. Clearly, $\Delta \geq 0$, or σ^* is not optimal. For these properties, we assume that E′ and T correspond to the nontardy and tardy jobs of σ^*, respectively.

Given an optimal schedule, the jobs in E′ and T must be chosen so that sliding all jobs earlier or later by ϵ does not reduce total cost. These requirements are formalized in Properties 3 and 4.

Property 3. *Given* σ^*, $\sum_{j\in E'} w_j \geq \sum_{j\in T} w_j$.

Proof. Suppose that we start all jobs earlier by ϵ where $0 < \epsilon < 1$. Then

$$\Delta = \epsilon\left(\sum_{j\in E'} w_j - \sum_{j\in T} w_j\right) \geq 0$$

$$\Rightarrow \sum_{j\in E'} w_j \geq \sum_{j\in T} w_j.$$

Property 4. *Given* σ^*, $\sum_{j\in T} w_j \geq \sum_{j\in E'} w_j - 2w_t$.

Proof. Suppose that we start all jobs later by ϵ where $0 < \epsilon < 1$. Then

$$\Delta = \epsilon\left[\sum_{j\in T} w_j + w_t - \left(\sum_{j\in E'} w_j - w_t\right)\right] \geq 0$$

$$\Rightarrow \sum_{j\in T} w_j \geq \sum_{j\in E'} w_j - 2w_t.$$

Given an optimal schedule, the jobs in E′ and T must be chosen so that moving the last job to the beginning of the schedule, or the first job to the end of the schedule, does not reduce total cost. These requirements are formalized in Properties 5 and 6.

Property 5. *Given* σ^*, $\sum_{j\in E'} p_j \geq \sum_{j\in T} p_j$.

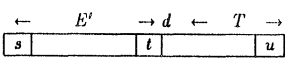

Figure 1. Schedule for optimality conditions.

Proof. Suppose that we place job u so that it is processed just before job s. Then

$$\Delta = w_u\left(\sum_{j\in E'} p_j - \sum_{j\in T} p_j\right) \geq 0$$

$$\Rightarrow \sum_{j\in E'} p_j \geq \sum_{j\in T} p_j.$$

Property 6. *Given* σ^*, $\sum_{j\in T} p_j \geq \sum_{j\in E'} p_j - 2p_s$.

Proof. Suppose that we place job s so that it is processed just after job u. Then

$$\Delta = w_s\left[\sum_{j\in T} p_j + p_s - \left(\sum_{j\in E'} p_j - p_s\right)\right] \geq 0$$

$$\Rightarrow \sum_{j\in T} p_j \geq \sum_{j\in E'} p_j - 2p_s.$$

The next two results specify conditions for a job to appear in a particular position in an optimal schedule. Property 7 gives a sufficient condition for job 1 to complete at d in an optimal schedule.

Property 7. If $p_1 = \max_j\{p_j\}$, *then there exists an optimal schedule where job* 1 *completes at d.*

Proof. Suppose that $1 \in E$ in some optimal schedule σ^*. By Property 2, $t = 1$.

When $1 \in T$, by Property 2, we can assume that job 1 starts at d. If jobs 1 and t are interchanged and no other job is moved, the change in cost is Δ where

$$\Delta = w_t p_1 + w_1(p_1 - p_t) - w_1 p_1$$

$$= w_t p_1 - w_1 p_t \leq 0.$$

Now $\{1, t\} \in T$. Since Property 4 holds for σ^*, this implies that for the current schedule, $\sum_{j\in E'} w_j \leq \sum_{j\in T} w_j$. Thus, we can slide all jobs earlier without increasing the cost until job 1 completes at d.

Property 8 gives a sufficient condition for a job to appear first in an optimal schedule.

Property 8. *If* $p_m = \max_j\{p_j\} \geq \sum_{j\neq m} p_j$, *then there exists an optimal schedule where job m is the first job processed.*

Proof. Suppose that $m \in T$ in some optimal schedule σ^*. Since $p_m = \max_j\{p_j\} \geq \sum_{j\neq m} p_j$, job m can be placed so that it is processed just before s, and the cost of the resulting schedule will not be greater. Therefore, we only need to consider the case when $m \in E'$ and m is not the first job. Let α be the set of jobs that precede m in σ^*. By moving the jobs in α to the end of σ^*, the cost associated with these jobs will not

increase. Thus, we have an optimal schedule that satisfies the conditions of Property 8.

2. NP-COMPLETENESS

The recognition version of **WET** is: Does there exist a nonpreemptive, single machine schedule with total weighted earliness and tardiness no larger than some integer y? To show that **WET** is NP-complete, we provide a reduction from the even-odd partition problem (Garey, Tarjan and Wilfong 1988), which is known to be NP-complete in the ordinary sense.

Even-Odd Partition. Given an integer n and a set $A = \{a_2, a_3, \ldots, a_{2n+1}\}$ of positive integers, where $a_i < a_{i+1}$ for $2 \leq i \leq 2n$, does there exist a partition of A into subsets A_1 and A_2, such that $\sum_{i \in A_1} a_i = \sum_{i \in A_2} a_i$, and that A_1 and A_2 each contains exactly one of $\{a_{2i}, a_{2i+1}\}$ for $1 \leq i \leq n$?

We show that answering the even-odd partition problem is equivalent to answering the recognition problem for **WET** for the instance $\mathbf{P_1}$ where there are $2n + 1$ jobs defined as follows.

Instance P_1

$$p_1 = 2S, \qquad w_1 = (n^2 + 2n + 1)S^2$$

$$p_{2j} = S + a_{2j+1}, \quad j = 1, 2, \ldots, n$$

$$p_{2j+1} = S + a_{2j}, \quad j = 1, 2, \ldots, n$$

$$w_j = S - a_j, \quad j = 2, 3, \ldots, 2n + 1$$

where

$$S = \left(\sum_{j=2}^{2n+1} a_j\right)^2,$$

and

$$y = S^2 n(n + 2) + S \sum_{k=1}^{n} (a_{2k} + a_{2k+1})(n - 2k) - \left(\sum_{j=2}^{2n+1} a_j\right)^2 \Big/ 4.$$

Notice that the weights and processing times are defined so that $w_1/p_1 \geq w_2/p_2 \geq \ldots \geq w_{2n+1}/p_{2n+1}$. Also, the weight of job $2i$ is matched with the processing time of job $2i + 1$ and the weight of job $2i + 1$ is matched with the processing time of job $2i$. This enables us to restore symmetry to the earliness-tardiness problem, which is inherently asymmetric. Problem **WET** is asymmetric in that if a job completes early, then its processing time does not contribute to its deviation from d. If a job is late, however, then its processing time is part of this deviation.

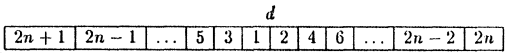

Figure 2. Schedule for Lemma 1.

The following definitions are used when we partition items of A into subsets E and T.

1. A partition is **a:b** if there are a items in E and b items in T.
2. A partition is **exact** if $\sum_{j \in E} a_j = \sum_{j \in T} a_j$.
3. A partition is **even–odd** if a_{2k} and a_{2k+1} are in different subsets for $1 \leq k \leq n$.

Any partition of the $2n$ items of the even–odd partition problem into subsets E and T corresponds to a schedule for **WET**, where jobs in E finish early (before d) and jobs in T finish late (after d). Job 1 has the largest processing time; by Property 7, in an optimal schedule it can finish processing exactly at d.

Lemma 1. *If there exists an exact n:n even–odd partition, then there exists a schedule σ for* $\mathbf{P_1}$ *where* $z(\sigma) = y$.

Proof. By assumption, an exact n:n even-odd partition exists. Define b_i, $i = 2, 3, \ldots, 2n + 1$, so that $\{b_{2i}, b_{2i+1}\} = \{a_{2i}, a_{2i+1}\}$ and $\sum_{k=1}^{n} b_{2k} = \sum_{k=1}^{n} b_{2k+1}$. This schedule, σ, is illustrated in Figure 2. Now,

$$\begin{aligned}
z(\sigma) &= (S - b_3)(2S) + (S - b_5)(3S + b_2) \\
&\quad + (S - b_7)(4S + b_2 + b_4) + \ldots \\
&\quad + (S - b_{2n+1})[(n + 1)S \\
&\qquad + b_2 + b_4 + \ldots + b_{2n-2}] \\
&\quad + (S - b_2)(S + b_3) + (S - b_4)(2S + b_3 + b_5) \\
&\quad + (S - b_6)(3S + b_3 + b_5 + b_7) + \ldots \\
&\quad + (S - b_{2n})[nS + b_3 + b_5 + \ldots + b_{2n+1}] \\
&= S^2[2 + 3 + \ldots + (n + 1) + 1 + 2 + \ldots + n] \\
&\quad + S[(n - 2)b_2 + (n - 2)b_3 + (n - 4)b_4 \\
&\qquad + (n - 4)b_5 + \ldots + (2 - n)b_{2n-2} \\
&\qquad + (2 - n)b_{2n-1} + (-n)b_{2n} + (-n)b_{2n+1}] \\
&\quad - [b_2(b_3 + b_5 + \ldots + b_{2n+1}) \\
&\qquad + b_4(b_3 + b_5 + \ldots b_{2n+1}) \\
&\qquad + \ldots + b_{2n}(b_3 + b_5 + \ldots + b_{2n+1})]
\end{aligned}$$

$$= S^2\left[\frac{(n+1)(n+2)}{2} - 1 + \frac{n(n+1)}{2}\right]$$
$$+ S\sum_{k=1}^{n}(b_{2k} + b_{2k+1})(n - 2k)$$
$$- \left(\sum_{k=1}^{n} b_{2k}\right)\left(\sum_{k=1}^{n} b_{2k+1}\right)$$
$$= S^2 n(n+2) + S\sum_{k=1}^{n}(b_{2k} + b_{2k+1})(n - 2k)$$
$$- \left(\sum_{j=2}^{2n+1} b_j\right)^2 \Big/ 4 = y$$

since the partition is exact.

For set K, let $|K|$ denote the cardinality of K.

Lemma 2. *$z(\sigma) \leqslant y$ only if $|\mathrm{E}| = n - 1$ and $|\mathrm{T}| = n + 1$, or if $|\mathrm{E}| = |\mathrm{T}| = n$.*

Proof. Since $S \gg \sum_{j=2}^{2n+1} a_j$, the S^2 term in the cost expression dominates the other terms. Any schedule σ with $|\mathrm{E}| = r$ and $|\mathrm{T}| = 2n - r$, has a cost given by

$$z(\sigma) = S^2[2 + 3 + \ldots + (r+1) + 1 + 2 + \ldots + (2n - r)] + O(S^{3/2})$$
$$= S^2[r^2 - r(2n-1) + n + 2n^2] + O(S^{3/2}).$$

This expression is a unimodal, strictly convex function of r, with a minimum value at $r = n - \frac{1}{2}$. There are two minimum integer values of r, i.e., $r = n$ and $r = n - 1$, where $z(\sigma) = S^2 n(n+2) + O(S^{3/2})$. It follows that only an $n - 1{:}n + 1$ or $n{:}n$ partition can correspond to a schedule with cost $\leqslant y$.

Lemma 3. *Among all $n - 1{:}n + 1$ or $n{:}n$ partitions, the schedule σ with the minimum cost corresponds to an $n{:}n$ even-odd partition.*

Proof. Suppose that we do not have an $n{:}n$ even–odd partition. Let i be the largest index such that jobs $2i$ and $2i + 1$ are both in E or in T. Let $\alpha_1 = \mathrm{E} \cap \{j \mid j \geqslant 2i + 2\}$ and $\beta_1 = \mathrm{T} \cap \{j \mid j \geqslant 2i + 2\}$. From the definition of i, $|\alpha_1| = |\beta_1|$. From Property 2, if jobs $2i$ and $2i + 1$ are both early or late in σ, they are adjacent.

We know from the proof of Lemma 2 that the S^2 term costs of all schedules corresponding to $n - 1{:}n + 1$ or $n{:}n$ partitions are equal. Since the S term dominates the terms that are independent of S, we only consider the S term costs. There are two cases, when jobs $2i$ and $2i + 1$ are both early and when they are both late.

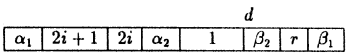

Figure 3. Schedule for Lemma 3, Case I.

Case I. Jobs $2i + 1$ and $2i$ are both early. In schedule σ, let $\alpha_2 = \mathrm{E} - \alpha_1 - \{2i + 1, 2i\}$, r be the last job processed before β_1, and $\beta_2 = \mathrm{T} - \beta_1 - \{r\}$. The schedule is shown in Figure 3. Since $|\alpha_1| = |\beta_1|$, $|\alpha_2| = |\beta_2| - 3$ if the partition is $n - 1{:}n + 1$, and $|\alpha_2| = |\beta_2| - 1$ if the partition is $n{:}n$. Observe that since $r < 2i$, $p_{2i} > p_r$ and $w_{2i} < w_r$. Let Δ_s denote the change in the S term of $z(\sigma)$ from interchanging jobs r and $2i$, and shifting the other jobs to maintain consecutive processing. Then

$$\Delta_s = (p_r - p_{2i})(|\alpha_1| + 1) + (w_r - w_{2i})(|\alpha_2| + 2) + (p_{2i} - p_r)|\beta_1| + (w_{2i} - w_r)(|\beta_2| + 1)$$
$$= (p_{2i} - p_r)(-1) + (w_{2i} - w_r)(|\beta_2| - |\alpha_2| - 1) < 0.$$

Therefore, the configuration where jobs $2i + 1$ and $2i$ are both early is not optimal.

Case II. Jobs $2i$ and $2i + 1$ are both late. Let k be the first job processed after α_1, $\alpha_2 = \mathrm{E} - \alpha_1 - \{k\}$ and $\beta_2 = \mathrm{T} - \beta_1 - \{2i, 2i + 1\}$. When $i > 1$, the schedule is shown in Figure 4. Here, $|\alpha_2| = |\beta_2| - 1$ if the partition is $n - 1{:}n + 1$, and $|\alpha_2| = |\beta_2| + 1$ if the partition is $n{:}n$. Observe that since $k < 2i$, $p_{2i} > p_k$ and $w_{2i} < w_k$. Let Δ_s denote the change in the S term of $z(\sigma)$ from interchanging jobs k and $2i$, and shifting the other jobs to maintain consecutive processing. Then

$$\Delta_s = (p_{2i} - p_k)|\alpha_1| + (w_{2i} - w_k)(|\alpha_2| + 2) + (p_k - p_{2i})(|\beta_1| + 1) + (w_k - w_{2i})(|\beta_2| + 1)$$
$$= (p_{2i} - p_k)(-1) + (w_{2i} - w_k)(|\alpha_2| - |\beta_2| + 1) < 0.$$

Therefore, the configuration where jobs $2i$ and $2i + 1$ are both late is not optimal.

When $i = 1$, the schedule is shown in Figure 5. This partition has the property of being even-odd for jobs $4, 5, \ldots, 2n + 1$. Thus, $|\mathrm{E}| = |\mathrm{T} - \{2, 3\}| = n - 1$ and the partition is $n - 1{:}n + 1$. Let Δ_s denote the change in the S term of $z(\sigma)$ from moving job 2 to a position immediately before job 1, and shifting the

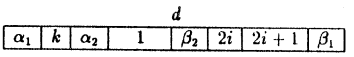

Figure 4. Schedule for Lemma 3, Case II, $i > 1$.

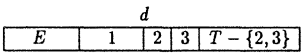

Figure 5. Schedule for Lemma 3, Case II, $i = 1$.

other jobs to maintain consecutive processing. Then

$$\Delta_s = p_2|\mathrm{E}| + 2w_2 - p_2(|\mathrm{T} - \{2, 3\}| + 2)$$
$$= 2(w_2 - p_2) < 0$$

so this $n - 1{:}n + 1$ partition is not optimal. From Cases **I** and **II**, if jobs $2i$ and $2i + 1$ are in the same half of the partition, then the schedule is not optimal.

Lemma 4. *Any $n{:}n$ even–odd partition that is not exact corresponds to a schedule σ, where $z(\sigma) > y$.*

Proof. From the proof of Lemma 1, the cost of an $n{:}n$ even–odd partition is

$$S^2n(n + 2) + S\sum_{k=1}^{n}(a_{2k} + a_{2k+1})(n - 2k)$$
$$-\left(\sum_{j\in\mathrm{E}} a_j\right)\left(\sum_{j\in\mathrm{T}} a_j\right) \geqslant y$$

where equality is attained only if $(\sum_{j\in\mathrm{E}} a_j)(\sum_{j\in\mathrm{T}} a_j) = (\sum_{j=2}^{2n+1} a_j)^2/4$, or equivalently, if $\sum_{j\in\mathrm{E}} a_j = \sum_{j\in\mathrm{T}} a_j = \sum_{j=2}^{2n+1} a_j/2$. This means that the partition is exact.

Theorem 1. *Problem* **WET** *is NP-complete in the ordinary sense.*

Proof. It can be seen that the recognition problem for **WET** is in NP, and that $\mathbf{P_1}$ can be constructed from the even–odd partition problem in polynomial time. We complete the proof by showing the equivalence of the even–odd partition problem and $\mathbf{P_1}$.

$\Rightarrow$ If there exists an $n{:}n$ even-odd exact partition, then there exists a solution to $\mathbf{P_1}$ with cost $= y$. This result is established in Lemma 1.

$\Leftarrow$ If there exists a solution to $\mathbf{P_1}$ with cost $\leqslant y$, there exists an $n{:}n$ even-odd exact partition.

Job 1 finishes processing exactly at d by Property 7. From Lemmas 2 and 3, if there exists a partition which corresponds to σ where $z(\sigma) \leqslant y$, then there exists an $n{:}n$ even–odd partition which does so. Finally, by Lemma 4, any $n{:}n$ even–odd partition which is not exact could not have cost $\leqslant y$. Thus, a positive response to the recognition question for **WET** is equivalent to a positive response for the even–odd partition problem.

As a result of this proof of NP-completeness, we do not expect to find an efficient (polynomial time) exact algorithm for **WET**.

3. A DYNAMIC PROGRAMMING ALGORITHM

We describe an effective dynamic programming algorithm for finding optimal solutions to **WET**. It is shown that the algorithm is pseudopolynomial, or polynomial with respect to a unary encoding of the data (see Papadimitriou and Steiglitz for details). Following the work of Lawler and Moore (1969), results of this kind have been demonstrated for a number of machine scheduling and other NP-complete problems. This result demonstrates that **WET** is not NP-complete in the strong sense. Thus, the proof of ordinary NP-completeness in Theorem 1, combined with this result, establishes a precise definition of the complexity of **WET**.

Before giving a formal description of our dynamic programming algorithm, we provide some insight into its operation. Using Property 1, some job completes at d. From Property 2, job n will be scheduled as either the first or the last job. Thus, the minimum cost of scheduling job n can be found if we know the total processing time of jobs previously scheduled early, and of jobs previously scheduled late. The identity of previously scheduled jobs $j \in \mathrm{E}'$ or $j \in \mathrm{T}$ is not relevant to the scheduling decision for job n. A similar insight applies to any other job, k, that is unscheduled. In fact, since $\sum_{j\in\mathrm{E}'\cap\{i<k\}} p_j + \sum_{j\in\mathrm{T}\cap\{i<k\}} p_j = \sum_{j=1}^{k-1} p_j$, it is necessary to know only the total processing time of jobs previously scheduled early. This insight suggests a dynamic programming procedure that uses a state variable to represent the total processing time of jobs previously scheduled early.

Algorithm Optwet

Let $f_k(e)$ = the minimum cost to schedule jobs k, $k + 1, \ldots, n$ given that, before job k is scheduled, the sum of processing times of jobs scheduled early (or on time) is e. The recurrence relation is:

$$f_k(e) = \min\left\{w_k e + f_{k+1}(e + p_k),\right.$$
$$\left. w_k\left(\sum_{j=1}^{k} p_j - e\right) + f_{k+1}(e)\right\}$$

and the boundary condition is:

$$f_{n+1}(e) = 0, \quad e = 0, 1, \ldots, \sum_{j=1}^{n} p_j.$$

The minimum cost schedule is found at $f_1(0)$. We note that the first and second alternative in the recurrence relation is the cost of scheduling job k early and late, respectively. To find the number of steps required by **Optwet**, note that k is $O(n)$ and e is $O(\sum_{j=1}^{n} p_j)$. Therefore, the recurrence relation is used $O(n \sum_{j=1}^{n} p_j)$ times. If we compute $\sum_{j=1}^{k} p_j$, $k = 2, 3, \ldots, n$, in $O(n)$ time prior to using the recurrence relation, each recurrence requires only constant time. Thus, the complexity of **Optwet** is $O(n \sum_{j=1}^{n} p_j)$. Consequently, **Optwet** is linear with respect to a unary encoding of the data, or is pseudopolynomial.

We can make the dynamic programming procedure more efficient through the observation that e can never exceed the total processing time of previously scheduled jobs, and the use of Properties 3–6. Specifically, at stage k, we compute $f_k(e)$, $e \in \{\underline{e}_k, \underline{e}_k + 1, \ldots, \bar{e}_k\}$, where we define $\underline{e}_k$ and $\bar{e}_k$ as follows. For stage k, $k = 1, 2, \ldots, n$, let:

$$r_1 = \min\left\{ i \;\middle|\; \left\lfloor \sum_{j=1}^{n} w_j/2 \right\rfloor \geq \sum_{j=i}^{k-1} w_j \right\} - 2,$$

$$r_2 = \min\left\{ i \;\middle|\; \left\lfloor \sum_{j=1}^{n} w_j/2 \right\rfloor + \max_{j=1,\ldots,k-1} \{w_j\} \geq \sum_{j=i}^{k-1} w_j \right\}.$$

Then:

$$\underline{e}_k = \max\left\{ 0, \sum_{j=1}^{r_1} p_j + \left\lceil \frac{p_{r_1+1}}{w_{r_1+1}} \left(\sum_{j=r_1+1}^{k-1} w_j - \left(\left\lfloor \sum_{j=1}^{n} w_j/2 \right\rfloor \right) \right) \right\rceil, \left\lceil \sum_{j=1}^{n} p_j/2 \right\rceil - \sum_{j=k}^{n} p_j \right\} \quad (1)$$

$$\bar{e}_k = \min\left\{ \sum_{j=1}^{k-1} p_j, \left\lfloor \sum_{j=1}^{n} p_j/2 \right\rfloor + \max_{j=1,\ldots,n} \{p_j\}, \sum_{j=r_2}^{k-1} p_j + \left\lfloor \frac{p_{r_2-1}}{w_{r_2-1}} \left(\left(\left\lfloor \sum_{j=1}^{n} w_j/2 \right\rfloor + \max_{j=1,\ldots,k-1} \{w_j\} - \sum_{j=r_2}^{k-1} w_j \right) \right) \right\rfloor \right\}. \quad (2)$$

Note that r_1 and r_2 follow from Properties 3 and 4, respectively. For a given k, the jobs with the smallest total processing time that could be processed after job k, complete by time d, and satisfy Property 3 are $\{1, 2, \ldots, r_1\}$ and a fraction of job $r_1 + 1$. This is the second term in (1). The third term in (1) is a direct result of Property 5. The jobs with the largest total processing time that could be processed after job k, complete by time d, and satisfy Property 4 are set $\{r_2, r_2 + 1, \ldots, k - 1\}$ and a fraction of job $r_2 - 1$. This is the third term of (2). The second term in (2) follows from Property 6. Additional strengthening of these bounds is possible. However, the increase in computational effort more than offsets the savings.

In order to test the computational effectiveness of **Optwet**, the algorithm was coded in FORTRAN on a Prime 9955 minicomputer and compiled using the F77 compiler. Altogether 150 random test problems were generated, consisting of 15 series of 10 problems each. Processing times and weights were independently generated from a uniform distribution over the integers 1, 2, ..., TOPSIZ, where TOPSIZ has values of 10, 20, 50 and 100. The results in Table I show the mean and the maximum solution times, over the 10 random problems in each series. Times are given in Prime 9955 CPU seconds, and do not include input and output time. The use of the bounds on $\underline{e}_k$ and $\bar{e}_k$ for $k = 1, 2, \ldots, n$, derived from Properties 3–6, considerably reduces the amount of computation needed. The reduction is about 50% for the problems tested, and does not vary significantly with the problem parameters.

The results suggest that our implementation of **Optwet**, combined with the conditions of Section 1, routinely solves to optimality large random instances of **WET** within about 60 seconds of minicomputer time.

From the prior discussion, the number of calculations required by **Optwet** should increase proportionally to $n(\sum_j p_j) \propto n^2\text{TOPSIZ}$. Initial investigations showed that the variance also grew at about the same rate. Consequently, we ran a weighted least squares

Table I
Computation Times for Random Problems

Series	n	TOPSIZ	Mean CPU Time (Sec.)	Maximum CPU Time (Sec.)
1	100	10	0.22	0.24
2	200	10	0.87	0.90
3	500	10	5.37	5.51
4	1,000	10	21.92	22.93
5	2,000	10	89.66	92.75
6	100	20	0.37	0.39
7	200	20	1.43	1.49
8	500	20	9.12	9.45
9	1,000	20	37.56	38.34
10	100	50	0.77	0.85
11	200	50	3.03	3.28
12	500	50	19.04	20.05
13	100	100	1.53	1.69
14	200	100	5.89	6.15
15	500	100	37.45	40.69

model adjusting for the heteroscedasticity of the data (see Tukey 1971, p. 108). The model is

$$\text{CPU Time} = \alpha + \beta_1 n + \beta_2 n^2 + \beta_3 n^2\text{TOPSIZ} + \epsilon,$$

$$\sigma^2(\text{CPU Time}) \propto n^2\text{TOPSIZ}$$

using all data. From a goodness-of-fit test, we were unable to reject H_0: *The residuals are normally distributed* at the 0.60 level. Test H_0: $\alpha = 0$ is rejected at the 0.00013 level, H_0: $\beta_1 = 0$ is rejected at less than the 0.00003 level, test H_0: $\beta_2 = 0$ is rejected at the 0.00079 level, and H_0: $\beta_3 = 0$ is rejected at less than the 0.00003 level. The coefficient of multiple determination is $R^2 = 0.946$. Therefore, the model appears to be a good predictor of CPU time.

4. A FULLY POLYNOMIAL APPROXIMATION SCHEME

In this section, we develop a fully polynomial approximation scheme for **WET** when the maximum weight is bounded by a polynomial function of n (see Papadimitriou and Steiglitz for definitions and references). Such schemes have the appealing property that they produce a heuristic solution with cost within a prespecified relative error of ϵ.

We use the dynamic programming procedure in Section 3. Let $p_m = \max_j\{p_j\}$ and $\hat{P} = \sum_j p_j - p_m$. Consider a modified problem where $w_j \leq g(n)$ for all j, $g(n)$ is a polynomial function of n, and

$$\tilde{p}_i = \left\lceil \frac{n^2 g(n) p_i}{\epsilon \hat{P}} \right\rceil, \quad i = 1, 2, \ldots, n.$$

To show that this procedure is polynomial in both n and $1/\epsilon$, if $p_m < \sum_j p_j/2$, then the dynamic programming procedure for the modified problem has complexity

$$O\left(n \sum_i \tilde{p}_i\right) = O\left(n\left(\sum_i \frac{n^2 g(n) p_i}{\epsilon \hat{P}} + 1\right)\right) = O\left(\frac{n^3 g(n)}{\epsilon}\right).$$

When $p_m \geq \sum_j p_j/2$, by Property 8, the job that corresponds to p_m is processed first, and the dynamic programming procedure has complexity

$$O\left(n\left(\sum_i \tilde{p}_i - \max\{\tilde{p}_i\}\right)\right) = O\left(\frac{n^3 g(n)}{\epsilon}\right).$$

Let $\tilde{\sigma}$ be the optimal schedule found for the modified problem and $\tilde{z}(\tilde{\sigma})$ be the associated cost. As ϵ goes to 0, $\tilde{z}(\tilde{\sigma})$ goes to ∞. As a result, we need to rescale the cost and let $\hat{z}(\tilde{\sigma}) = \tilde{z}(\tilde{\sigma})\epsilon\hat{P}/n^2 g(n)$. Also, let $\sigma^* = (u_1, u_2, \ldots, u_q@, u_{q+1}, \ldots, u_n)$ be an optimal schedule for the original problem. Because $p_i \leq \tilde{p}_i \epsilon \hat{P}/n^2 g(n)$ for all i, $\hat{z}(\tilde{\sigma}) \geq z(\tilde{\sigma}) \geq z(\sigma^*)$. If R is the relative error, and $w_j \leq g(n)$ for all j, then

$$R = \frac{z(\tilde{\sigma}) - z(\sigma^*)}{z(\sigma^*)} \leq \frac{\hat{z}(\tilde{\sigma}) - z(\sigma^*)}{z(\sigma^*)} \leq \frac{\hat{z}(\sigma^*) - z(\sigma^*)}{z(\sigma^*)}$$

$$= \left\{ \sum_{k=1}^{q-1} w_{u_{q-k}} \sum_{i=1}^{k} \left(\frac{\epsilon\hat{P}}{n^2 g(n)} \left\lceil \frac{n^2 g(n) p_{u_{q-i+1}}}{\epsilon\hat{P}} \right\rceil - p_{u_{q-i+1}} \right) \right.$$

$$\left. + \sum_{k=q+1}^{n} w_{u_k} \sum_{i=q+1}^{k} \left(\frac{\epsilon\hat{P}}{n^2 g(n)} \left\lceil \frac{n^2 g(n) p_{u_i}}{\epsilon\hat{P}} \right\rceil - p_{u_i} \right) \right\} \Big/ z(\sigma^*)$$

$$\leq \frac{\sum_{k=1}^{q-1} w_{u_{q-k}} \sum_{i=1}^{k} \frac{\epsilon\hat{P}}{n^2 g(n)} + \sum_{k=q+1}^{n} w_{u_k} \sum_{i=q+1}^{k} \frac{\epsilon\hat{P}}{n^2 g(n)}}{\sum_{k=1}^{q-1} w_{u_{q-k}} \sum_{i=1}^{k} p_{u_{q-i+1}} + \sum_{k=q+1}^{n} w_{u_k} \sum_{i=q+1}^{k} p_{u_i}}$$

$$\leq \frac{\frac{\epsilon\hat{P}}{n^2 g(n)} \left(\sum_{k=1}^{q-1} w_{u_{q-k}} \sum_{i=1}^{k} 1 + \sum_{k=q+1}^{n} w_{u_k} \sum_{i=q+1}^{k} 1 \right)}{\sum_{k=1}^{q-1} \sum_{i=1}^{k} p_{u_{q-i+1}} + \sum_{k=q+1}^{n} \sum_{i=q+1}^{k} p_{u_i}}$$

$$\leq \frac{\epsilon\hat{P}}{\sum_{k=1}^{q-1} \sum_{i=1}^{k} p_{u_{q-i+1}} + \sum_{k=q+1}^{n} \sum_{i=q+1}^{k} p_{u_i}} \leq \epsilon.$$

This implies that we have a fully polynomial approximation scheme.

5. CASES WHEN WET IS POLYNOMIAL

In this section, we consider four special cases of **WET** for which an optimal solution can be found in polynomial time. Recall that Kanet provides an efficient algorithm for the case when $w_j = 1, j = 1, 2, \ldots, n$.

5.1. Weights and Processing Times Are Equal

Assume that $w_j = p_j, j = 1, 2, \ldots, n$. This assumption conforms naturally to many practical situations in which value accrues at a constant rate over time. For this case, we describe an algorithm, **Bigfirst**, requiring only $O(n)$ steps. Informally, **Bigfirst** takes jobs in nonincreasing size order and schedules them to finish early, until the total processing time of jobs scheduled is at least half of the overall total processing time. All other jobs are scheduled late. Note that the order of the jobs in E′ or T does not affect cost because $w_j/p_j = 1, j = 1, 2, \ldots, n$.

Algorithm Bigfirst

Step 0. We are given n arbitrarily ordered jobs with processing times $p_1, p_2, \ldots, p_n$ and weights $w_1, w_2, \ldots, w_n$, where $w_j = p_j$, $j = 1, 2, \ldots, n$. Let $E' = \varnothing$ (jobs scheduled to complete processing by d), $e = 0$ (total processing time of jobs scheduled in E'), $Q = \{1, 2, \ldots, n\}$ (set of unscheduled jobs), and $\bar{P} = \lceil \sum_{i=1}^{n} p_i/2 \rceil$.

Step 1. Find the median value q of $\{p_i \mid i \in Q\}$.

Step 2. Let $I = \{i \in Q \mid p_i = q\}$. Determine $H = \{i \in Q \mid p_i > q\} \cup I_1$ such that $I_1 \subseteq I$ and

$$|H| = \lceil |Q|/2 \rceil.$$

Step 3. If $\sum_{i \in H} p_i + e \geq \bar{P}$, then $Q \leftarrow H$. Otherwise, $E' \leftarrow E' \cup H$, $e \leftarrow e + \sum_{i \in H} p_i$, and $Q \leftarrow Q \backslash H$.

Step 4. If $Q \neq \varnothing$, then go to Step 1.

Step 5. $T = \{1, 2, \ldots, n\} \backslash E'$. Stop.

To determine the computational complexity of **Bigfirst**, notice that Step 1 can be completed in $O(|Q|)$ time by using a linear time method to find the median (see Blum et al. 1973, and Schönhage, Paterson and Pippenger 1976). Steps 2 and 3 are also $O(|Q|)$. Step 5 is $O(n)$ but is performed only once. Thus, each iteration of **Bigfirst** is $O(|Q|)$. Each use of Step 3 reduces the cardinality of Q to $\lceil |Q|/2 \rceil$ or less. Hence, the order of **Bigfirst** is

$$O\left(n + \left\lceil \frac{n}{2} \right\rceil + \left\lceil \frac{\lceil n/2 \rceil}{2} \right\rceil + \cdots + 1\right)$$

$$= O\left(n + \frac{n+1}{2} + \frac{n+3}{4} + \frac{n+7}{8} + \cdots + 1\right)$$

$$= O\left(n + \frac{n}{2} + \frac{n}{4} + \cdots + \frac{n}{2^{\lceil \log_2 n \rceil}}\right) + O(\log_2 n)$$

$$= O(n).$$

To show that **Bigfirst** delivers an optimal solution, we first show that the big jobs go in E'.

Lemma 5. *There exists an optimal schedule σ^*, where if* E' *and* T *correspond to σ^*, then for all jobs $i \in E'$ and $j \in T$, $p_i \geq p_j$.*

Proof. Suppose there exist jobs $t \in E'$ and $j \in T$ in σ^* such that $p_t < p_j$. Since the order of jobs within E' and T does not affect cost, we can assume without loss of generality that job t completes processing at d, and job j starts processing at d. Let Δ denote the change in cost from interchanging jobs t and j so that j completes at d, t starts at d, and the other jobs are shifted to maintain consecutive processing. Then

$$\Delta = (p_j - p_t)\left(\sum_{k \in E'} w_k - w_t\right) + w_t p_t - w_j p_j + (p_t - p_j)\left(\sum_{k \in T} w_k - w_j\right)$$

$$= (p_j - p_t)\left(\sum_{k \in E'} p_k - p_t\right) + p_t^2 - p_j^2 + (p_t - p_j) \sum_{k \in T} (p_k - p_j)$$

$$= (p_j - p_t)\left(\sum_{k \in E'} p_k - \sum_{k \in T} p_k - 2p_t\right).$$

Suppose that $\sum_{k \in E'} p_k > \sum_{k \in T} p_k + 2p_t$. Then using $w_j = p_j$, $j = 1, 2, \ldots, n$, $\sum_{k \in E'} w_k > \sum_{k \in T} w_k + 2w_t$, and Property 4 is not satisfied. Thus, we can assume that $\sum_{k \in E'} p_k \leq \sum_{k \in T} p_k + 2p_t$. Since it follows that $\Delta \leq 0$, the schedule can be replaced by one with a cost that is not larger, and $j \in E'$ and $t \in T$. This procedure can be repeated until the conditions of the lemma are satisfied.

Theorem 2. *Algorithm* **Bigfirst** *delivers an optimal schedule to problem* **WET** *with $w_j = p_j$, $j = 1, 2, \ldots, n$.*

Proof. From Lemma 5, **Bigfirst** delivers an optimal schedule with $E' = \{1, 2, \ldots, k\}$ and $T = \{k+1, k+2, \ldots, n\}$ provided that the stopping condition for choosing E' in Step 1 is correct. If $e < \sum_{j=1}^{n} p_j/2$, then $e = \sum_{j \in E'} p_j < \sum_{j \in T} p_j$, and Property 5 does not hold. If $e > p_k + \sum_{j=1}^{n} p_j/2$, then $\sum_{j \in T} p_j < \sum_{j \in E'} p_j - 2p_k$, and Property 6 does not hold. In the case when $e = p_k + \sum_{j=1}^{n} p_j/2$, sets $E_1' = \{1, 2, \ldots, k-1\}$ and $E_2' = \{1, 2, \ldots, k\}$ give the same cost schedule. Thus, the E' and T that are generated by **Bigfirst** give an optimal schedule.

Note that **Bigfirst** does not necessarily provide good results for the general version of **WET**. The following example shows that **Bigfirst** can have an arbitrarily bad worst case performance. Let $n = 2$, $p_1 = 2k$, $w_1 = 1$, $p_2 = w_2 = k$. The **Bigfirst** schedule is (1@, 2) with a cost of k^2, while the optimal schedule is (2@, 1) with a cost of $2k$.

5.2. Jobs Have Unit Processing Times

For this class of problems, we assume that $p_j = 1$ for all j. From Properties 1 and 2, the completion times of the jobs are $d - \lceil n/2 \rceil + 1$, $d - \lceil n/2 \rceil + 2$, ...,

$d - \lceil n/2 \rceil + n$. From simple cost considerations, an optimal solution σ^* is $(n-1, n-3, \ldots, 3, 1@, 2, 4, \ldots, n)$ if n is even, and $(n, n-2, \ldots, 3, 1@, 2, 4, \ldots, n-1)$ if n is odd. In either case,

$$z(\sigma^*) = \sum_{j=1}^{n} \lfloor j/2 \rfloor w_j.$$

5.3. No Job Is Late

For this class of problems, we generate sufficient conditions for all jobs to be in E′.

Theorem 3. *If* $p_k \geq 2 \sum_{i=1}^{k-1} p_i$ *and* $w_k \geq 2 \sum_{i=k+1}^{n} w_i$ *for* $k = 1, 2, \ldots, n$, *then the schedule* $(n, n-1, \ldots, 2, 1@)$ *is optimal.*

Proof. Suppose that for some job k, $k \in \mathrm{T}$. Let Δ denote the change in cost from inserting job k in its appropriate position in E′. Then

$$\Delta \leq \left(\sum_{i=k+1}^{n} w_i \right) p_k + w_k \sum_{i=1}^{k-1} p_i - w_k p_k$$

$$\leq w_k p_k/2 + w_k p_k/2 - w_k p_k = 0.$$

If $|\mathrm{T}| > 1$, the same operation can be repeated until T is empty.

This result shows the inherent asymmetry in **WET**.

5.4. The First Job Is Large

The fourth class of problems where it is easy to determine an optimal solution is when there exists one very large and important (large weight) job that dominates processing. In this case only one job is in E′.

Theorem 4. *If* $p_1 \geq \sum_{i=2}^{n} p_i$, *then the schedule* $(1@, 2, 3, \ldots, n)$ *is optimal.*

Proof. From Property 7, job 1 completes at d. As a result, $\mathrm{E}' = \{1\}$ by Property 8. Finally, from Property 6, $\mathrm{T} = \{2, 3, \ldots, n\}$.

6. FINAL REMARKS

The weighted earliness-tardiness problem has been shown to be hard in a complexity sense, but tractable from a computational standpoint. Polynomial time algorithms have been provided for four special cases of the problem. While several closely related problems have been discussed in the literature, there is no similar problem which has been shown to be NP-complete, but also to admit a pseudopolynomial algorithm. Thus, the boundary between easy and hard problems has been further narrowed. Our computational results suggest that large instances of the problem can be solved to optimality, obviating the need for heuristic approaches in most applications.

We conjecture that it is not possible to find a fully polynomial approximation scheme for **WET** when the weights are not bounded. If the weights are an explicit part of the procedure, then they will appear in the order of complexity, and the procedure is not polynomial. If the weights are not an explicit part, then $[z(\bar{\sigma}) - z(\sigma^*)]/z(\sigma^*)$ can be made arbitrarily large by carefully selecting a subset of the weights to increase.

A number of closely related problems remain as topics for future investigations. For instance, it would be interesting to know which cases in Section 5 remain solvable if the weights are asymmetric, that is, have different values for earliness and tardiness. It follows immediately from our results that the weighted earliness-tardiness problem with arbitrary common due date (i.e., d is not necessarily $\geq \sum_{j=1}^{n} p_j$) is NP-complete. The complexity of the unweighted version of this problem is resolved in Part II, the companion paper by Hall, Kubiak and Sethi (1991).

ACKNOWLEDGMENT

We thank Jack Neuhardt for his assistance with the analysis of the data in Section 3. We also thank Jay Ghosh for a helpful comment regarding Section 4. Two referees provided comments that improved the presentation of the paper.

REFERENCES

ACHUTAN, N. R., J. GRABOWSKI AND J. B. SIDNEY. 1981. Optimal Flow-Shop Scheduling With Earliness and Tardiness Penalties. *Opsearch* **18,** 117–138.

BAGCHI, U. 1989. Simultaneous Minimization of Mean and Variation of Flow Time and Waiting Time in Single Machine Systems. *Opns. Res.* **37,** 118–125.

BAGCHI, U., Y. L. CHANG AND R. S. SULLIVAN. 1987. Minimizing Absolute and Squared Deviations of Completion Times With Different Earliness and Tardiness Penalties and a Common Due Date. *Naval Res. Logis.* **34,** 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 Res. Logis. Quart.* **33,** 227–240.

BAGCHI, U., R. S. SULLIVAN AND Y. L. CHANG. 1987. Minimizing Mean Squared Deviation of Completion Times About a Common Due Date. *Mgmt. Sci.* **33,** 894–906.

BAKER, K. R., AND G. D. SCUDDER. 1990. Sequencing With Earliness and Tardiness Penalties: A Review. *Opns. Res.* **38,** 22–36.

BLUM, N., R. W. FLOYD, V. PRATT, R. L. RIVEST AND R. E. TARJAN. 1973. Time Bounds for Selection. *J. Comput. Syst. Sci.* **7,** 448–461.

EILON, S., AND I. G. CHOWDHURY. 1977. Minimizing Waiting Time Variance in the Single Machine Problem. *Mgmt. Sci.* **23,** 567–575.

EMMONS, H. 1987. Scheduling to a Common Due Date on Parallel Uniform Processors. *Naval Res. Logis.* **34,** 803–810.

GAREY, M. R., AND D. S. JOHNSON. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York.

GAREY, M. R., R. E. TARJAN AND G. T. WILFONG. 1988. One-Processor Scheduling With Symmetric Earliness and Tardiness Penalties. *Math. Opns. Res.* **13,** 330–348.

HALL, N. G. 1986. Single and Multiple Processor Models for Minimizing Completion Time Variance. *Naval Res. Logis. Quart.* **33,** 49–54.

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. *Opns. Res.* **39,** 847–856.

HENDERSON, B. 1982. Perspectives—Just In Time (JIT). Boston Consulting Group, Boston.

KANET, J. J. 1981. Minimizing the Average Deviation of Job Completion Times About a Common Due Date. *Naval Res. Logis. Quart.* **28,** 643–651.

KUBIAK, W., S. LOU AND S. P. SETHI. 1990. Equivalence of Mean Flow Time Problems and Mean Absolute Deviation Problems. *Opns. Res. Letts.* **9,** 371–374.

LAKSHMINARAYAN, S., R. LAKSHMANAN, R. L. PAPINEAU AND R. ROCHETTE. 1978. Optimal Single-Machine Scheduling With Earliness and Tardiness Penalties. *Opns. Res.* **26,** 1079–1082.

LAWLER, E. L., AND J. M. MOORE. 1969. A Functional Equation and Its Application to Resource Allocation and Sequencing Problems. *Mgmt. Sci.* **16,** 77–84.

LENSTRA, J. K., A. H. G. RINNOOY KAN AND P. BRUCKER. 1977. Complexity of Machine Scheduling Problems. *Annals Discrete Math.* **1,** 343–362.

OW, P. S., AND T. E. MORTON. 1989. The Single Machine Early/Tardy Problem. *Mgmt. Sci.* **35,** 177–191.

PAPADIMITRIOU, C. H., AND K. STEIGLITZ. 1982. *Combinatorial Optimization.* Prentice-Hall, Englewood Cliffs, N.J.

PORTEUS, E. L. 1985. Investing in Reduced Setups in the EOQ Model. *Mgmt. Sci.* **31,** 998–1010.

SCHÖNHAGE, A., M. PATTERSON AND N. PIPPENGER. 1976. Finding the Median. *J. Comput. Syst. Sci.* **13,** 184–199.

SIDNEY, J. B. 1977. Optimal Single-Machine Scheduling With Earliness and Tardiness Penalties. *Opns. Res.* **25,** 62–69.

SUNDARARAGHAVAN, P. S., AND M. U. AHMED. 1984. Minimizing the Sum of Absolute Lateness in Single-Machine and Multimachine Scheduling. *Naval Res. Logis. Quart.* **31,** 325–333.

SZWARC, W. 1989. Single Machine Scheduling to Minimize Absolute Deviation of Completion Times From a Common Due Date. *Naval Res. Logis.* **5,** 663–674.

TUKEY, J. W. 1971. *Exploratory Data Analysis.* Addison-Wesley, Reading, Mass.