REPRINTED FROM:

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

European Journal of Operational Research 70 (1993) 272–288
North-Holland

On scheduling to minimize earliness–tardiness and batch delivery costs with a common due date

Jeffrey W. Herrmann and Chung-Yee Lee
Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL 32611, USA

Received December 1991; revised September 1992

NORTH-HOLLAND · AMSTERDAM–LONDON–NEW YORK–TOKYO

European Journal of Operational Research 70 (1993) 272–288
North-Holland

On scheduling to minimize earliness–tardiness and batch delivery costs with a common due date

Jeffrey W. Herrmann and Chung-Yee Lee

Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL 32611, USA

Received December 1991; revised September 1992

Abstract: In this paper, we consider a single-machine scheduling problem where all jobs have a common due date. The problem is to minimize the sum of earliness and tardiness penalties and the delivery costs of the tardy jobs, where the tardy jobs are delivered in batches with a fixed cost per batch. Our approach is to use a pseudo-polynomial dynamic programming algorithm to solve the problem. We also discuss some special cases that are solvable in polynomial time and show that for a given schedule of tardy jobs, the problem of scheduling the batch deliveries is equivalent to the dynamic lot sizing problem. We describe how the general problem is much more difficult. Finally, we present the results of empirical testing of the dynamic program and a number of heuristics developed.

Keywords: Single-machine scheduling; Batch delivery; Common due date

1. Introduction

The importance of just-in-time (JIT) manufacturing to industry has led to the investigation of scheduling problems that include both earliness and tardiness penalties. Customers operating in a JIT environment do not want their orders delivered early, and thus a shop that finishes a job early must store the completed material until the due date, incurring holding costs as an earliness penalty. However, jobs delivered after the due data incur some tardiness cost dictated by the customer or associated with the loss of customer goodwill. This type of earliness–tardiness scheduling is different from traditional scheduling problems and has attracted much attention in the last decade.

For a survey of the literature on earliness-tardiness problems, see Baker and Scudder (1990). Recent papers in the field include Hall and Posner (1991), Hall, Kubiak and Sethi (1991), Liman and Lee (1992), and Federgruen and Mosheiov (1991). Most researchers have ignored shipping costs as either zero or irrelevant. However, it is well known in industry that shipping costs are a significant factor in the inventory operating cost. A more realistic production model should include both the sequencing of the jobs and the scheduling of the deliveries.

Cheng and Kahlbacher (1991) study an earliness problem with job batching and batch delivery costs. Lee, Danusaputro and Lin (1991) study the problem where jobs that finish early incur a holding cost until delivered at the common due date and jobs that finish tardy have a linear tardiness penalty and a fixed cost for each tardy job. This fixed cost includes not only a set tardy penalty cost but also a shipping cost.

Correspondence to: Dr. Chung-Yee Lee, Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL 32611, USA.

Early jobs are delivered on-time for a cost that is independent of the number of early jobs and thus can be ignored. The authors implicitly assume that a tardy job is always delivered as soon as it is completed.

This paper investigates a single-machine, deterministic, common due date scheduling problem with the sum of earliness penalties, tardiness penalties, and shipping costs as the objective function. We assume that the cost per tardy delivery is fixed, that is, independent of the jobs delivered at this time. Hence it may be cheaper to delay shipping a job until the delivery time of a following job, since this delay saves a delivery charge, although it incurs additional holding (earliness) and tardiness costs. Namely, a job that finishes tardy may incur both earliness and tardiness penalties in order to avoid a shipping cost. Hence the problem involves the sequencing of the jobs as well as the scheduling of the deliveries. The problem of scheduling these deliveries can be considered as a batch-flow problem. Such batching processes have been studied by Dobson, Karmarkar and Rummel (1987) with the assumption that the work is continuously divisible.

In this paper, we formulate the general problem and introduce notation, derive some basic results, and present a dynamic programming approach to solve the problem. In addition, we discuss two special cases that can be solved in polynomial time and describe how the problem is similar to a dynamic lot sizing model. We consider the general problem in order to show how much more difficult it is. We present the results of empirical tests of the dynamic program and discuss a number of simple heuristics developed for the problem.

2. Notation and some properties

We will first introduce some notation: For each job J_j, $j = 1, \ldots, n$, the following data are given:

p_j = Processing time.
α_j = Earliness penalty rate.
β_j = Tardiness penalty rate.
K = Batch delivery cost.
d = The given common due date.

We assume that d is a given parameter and restrictive, $d \leq p_1 + \cdots + p_n$. Thus this due date restricts the set of jobs that can be delivered on-time.

For a given schedule with tardy batch deliveries, let

C_j = The completion time of J_j.
D_j = The delivery date of J_j, where $D_j \geq C_j$ and $D_j = d$ if $C_j \leq d$.
T_j = The tardiness of $J_j = D_j - d$.
E_j = The earliness of $J_j = D_j - C_j$.

It can be shown easily that if J_j completes after the due date ($C_j > d$), we should deliver J_j either at its completion time C_j ($D_j = C_j$) or at the completion time C_i of some other job J_i ($D_j = C_i$, where $C_i > C_j$). Hence we can formulate the general problem as

$$\text{Min} \quad \sum (\alpha_j E_j + \beta_j T_j + K y_j)$$

where $y_j = 1$ if $D_j = C_j > d$ and 0 otherwise.

Note that even if $K = 0$ and $\alpha_j = \beta_j$ for all J_j, the problem is NP-complete, as proved by Hall and Posner (1991). Hence our general problem is NP-complete. It is unlikely that there exists an efficient polynomial algorithm to solve the problem optimally, and the use of a pseudo-polynomial algorithm can be justified.

Before we provide a pseudo-polynomial dynamic programming to solve the problem, first note that if the tardy batch delivery cost K is small enough, then, as the following theorem shows, each tardy job should be delivered as it completes. Hence the problem can be solved by the algorithm in Lee, Danusaputro and Lin (1991) if the jobs have agreeable ratios, that is $p_i/\alpha_i < p_j/\alpha_j$ implies $p_i/\beta_i \leq p_j/\beta_j$.

Theorem 1. *If $K < \alpha_i p_j + \beta_j p_i$ for all J_i and J_j, then for all tardy jobs, the optimal policy is to ship each one as it finishes.*

Proof. Consider a sequence where a tardy batch has more than one job. Let A be the jobs in one such tardy batch and J_j the last job in the batch. Now, remove J_j from A and schedule J_j (with a new delivery) before A' (the jobs that remain from A). The added cost due to the new delivery for J_j is K. The savings due to reduced tardiness of J_j is $\beta_j p(A')$, where $p(A')$ is the sum of processing times of the jobs in A'. The savings due to reduced earliness of jobs in A' is $\alpha(A')p_j$, where $\alpha(A')$ is the sum of the unit earliness penalties of the jobs in A'. Let J_k be any job in A'. Then, $p(A') \geq p_k$ and $\alpha(A') \geq \alpha_k$. The total cost change is $K - \beta_j p(A') - \alpha(A')p_j \leq K - \beta_j p_k - \alpha_k p_j$. This quantity, by the given, is negative. Thus, we can reduce the total cost by removing a job from a multi-job batch and delivering it separately. □

Although we will return to it later, for now we will set aside the general (weighted) problem and consider the simpler, equally-weighted case. We will concentrate on the problem where $\alpha_j = \alpha$ and $\beta_j = \beta$ for all $j = 1, \ldots, n$, and $\alpha \leq \beta$. The assumption $\alpha \leq \beta$ seems reasonable since tardiness affects the customer, who should be more important. This problem is NP-complete since Hall, Kubiak and Sethi (1991) prove that the special case where $K = 0$, $\alpha = 1$, and $\beta = 1$, is NP-complete. This provides the justification for developing a pseudo-polynomial algorithm to solve the problem. In the following section we describe the dynamic programming algorithm we developed.

3. Dynamic programming algorithm

Before we develop a dynamic program to solve the problem, we first discuss a number of properties about optimal schedules for our problem.

For previously-studied common due date earliness–tardiness problems without shipping costs, the properties of an optimal schedule are known:

I. The schedule is continuous (there exists no inserted idle time).
II. The schedule is V-shaped: the early jobs are arranged by longest processing time first (LPT), the tardy jobs by shortest processing time first (SPT).
III. Either there exists a job in the schedule that ends at the due date, or the schedule starts at time zero.

The first two properties are well-known; see Hall and Posner (1991), for instance. Hall, Kubiak and Sethi (1991) prove the third property, a combination of the properties for the restrictive and unrestrictive cases.

For the problem studied in this paper, properties I and III will hold (the proof of I is trivial, and the proof of III is the same as that in Hall, Kubiak and Sethi), although property II will not. This is due to the fact that the delivery time of a job is not necessarily its completion time.

An optimal feasible schedule for the problem consists of a number of early jobs that are delivered at the common due date. Following the early jobs are those jobs that finish tardy and are delivered in batches. Each batch is a set of consecutive jobs and it is easy to see that the delivery of a batch should occur at the completion time of the last job in the batch. Now, note that the jobs in a batch all have the same tardiness because they are delivered together. They also incur earliness costs waiting for the later jobs in the batch to finish. Thus, the set of jobs in a batch should be ordered by LPT to minimize that waiting cost.

Let us define S_k as the set of tardy jobs delivered at the k-th tardy delivery.

Lemma 1. *In an optimal schedule, for any k, the jobs in S_k are ordered by* LPT.

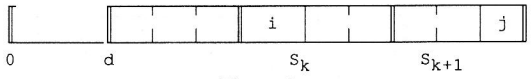

Figure 1

Proof. Suppose not. Then there exists J_i, J_j, and $S_k : J_i$ immediately precedes J_j, $p_i < p_j$, and both J_i and J_j are in S_k. If we interchange J_i and J_j, their tardiness penalties stay the same. The earliness of J_i decreases by p_j and that of J_j increases by p_i. The net change in cost is therefore $\alpha(p_i - p_j)$. This change is negative, and thus our new schedule is better than our optimal schedule, which is a contradiction. □

In the problem without shipping costs, the tardy jobs would be sequenced in SPT order. It is interesting to see that in our problem the tardy batches may maintain some form of this SPT property.

Lemma 2. *There exists an optimal schedule such that for all $k > 0$, if J_i is in S_k and J_j is in S_{k+1}, $p_i \le p_j$. We call this property* batch-SPT.

Proof. Because of Lemma 1, the condition of Lemma 2 is true for an optimal schedule if and only if, for all k, $p_i \le p_j$ where J_i is the first job of S_k and J_j is the last job of S_{k+1}. Call this statement T. Now, suppose that there exists an optimal schedule such that for some k and appropriate J_i and J_j, $p_i > p_j$. Let B be the number of jobs in S_k and let A be the number of jobs preceding J_j in S_{k+1}. Thus S_{k+1} has a total of $A + 1$ jobs (see figure 1).

If $A < B$, swap J_i and J_j. The earliness and tardiness penalties of all jobs not in S_k or S_{k+1} are unchanged. The tardiness of each of the A jobs in S_{k+1} before J_i is the same, but the earliness of each job is increased by $p_i - p_j$. J_i takes up the tardiness J_j lost. For the jobs now in S_k, the earliness of each job is the same and J_j now has the earliness J_i lost. The tardiness of each of the B jobs now in S_k is decreased by $p_i - p_j$. The net change in cost is $A\alpha(p_i - p_j) - B\beta(p_i - p_j) = (A\alpha - B\beta)(p_i - p_j)$. Because $\alpha \le \beta$ and $A < B$, $A\alpha < B\beta$, and the net change is negative.

If $A \ge B$, place J_j in the last position of S_k (see Figure 2). Again, the earliness and tardiness penalties of all jobs not in S_k or S_{k+1} are unchanged. Let P be the sum of the processing times of the $A + 1$ jobs that were in S_{k+1}. For S_k, each of the B jobs has its earliness increased by p_j and its tardiness increased by p_j. The job J_j gains no earliness but does lose $P - p_j$ in tardiness. Each of the A remaining jobs in S_{k+1} lose p_j in earliness. The total change in cost is $Bp_j(\alpha + \beta) - \beta(P - p_j) - A\alpha p_j$. Now, note that by LPT, J_j is the shortest job in S_{k+1} and thus $P \ge (A + 1)p_j$. Thus the change in cost is bounded above by $Bp_j(\alpha + \beta) - \beta Ap_j - A\alpha p_j$, which equals $(B - A)p_j(\alpha + \beta)$. Because $A \ge B$, this term is less than or equal to zero, and our net change is not positive.

After the interchange, rearrange the jobs within each of S_k and S_{k+1} by LPT, which cannot increase the objective function (by Lemma 1). Repeat these steps until statement T is true. Then, we have a schedule that is no worse than our original and is thus also optimal. □

We now know that there exists an optimal schedule for our problem with the following three properties (note that Property II′ replaces the Property II that holds for the problem without delivery costs):

I. The schedule is continuous (there exists no inserted idle time).

II′. The early jobs and the jobs in any batch are sequenced by LPT, and the tardy jobs are in batch-SPT order

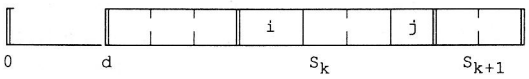

Figure 2

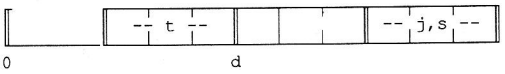

Figure 3

III. Either there exists a job in the schedule that ends at the due date, or the schedule starts at time zero.

In order to cover the two cases mentioned in Property III, we now present two dynamic programs to solve this problem. The first dynamic program (Algorithm 1a) solves the case where a job finishes at the common due date; the second dynamic program (Algorithm 1b) solves the case where the first job must start at time zero. Both dynamic programs make use of the orderings implied by Property II′. The solution to the problem is the best of the two solutions obtained from the dynamic programs.

Algorithm 1a. Renumber the jobs such that $p_i \le p_{i+1}$ for all i. Let $P_i = p_1 + p_2 + \cdots + p_i$. Let $c(i, j, s, t)$ be the minimum cost of scheduling the i shortest jobs with j jobs in the last tardy batch where those jobs have a combined processing time of s and the combined processing times of the early jobs is t. (All jobs are pushed to the due date. See Figure 3.) $i = 0, \ldots, n$. $j = 0, \ldots, n$. $s = 0, \ldots, P_n$. $t = 0, \ldots, d$.

Initial values: $c(0, 0, 0, 0) = 0$.
$c(0, j, s, t) =$ infinity otherwise.
$c(i, j, s, t) =$ infinity for $j < 0$, $s < 0$, or $t < 0$.

Iteration:
if $j \neq 1$ or $s \neq p_i$,

$$c(i, j, s, t) = \min \begin{cases} c(i-1, j, s, t-p_i) + (t-p_i)\alpha, \\ c(i-1, j-1, s-p_i, t) + (P_i - t)\beta + (j-1)p_i\beta + (s-p_i)\alpha, \end{cases}$$

if $j = 1$ and $s = p_i$,

$$c(i, j, s, t) = \min \begin{cases} c(i-1, j, s, t-p_i) + (t-p_i)\alpha, \\ c^*(i-1, t) + (P_i - t)\beta + K, \end{cases}$$

where $c^*(i-1, t) = \min\{c(i-1, j, s, t) \text{ for all } j, s\}$.
Update $c^*(i, t)$ if necessary.
Answer: minimum $c(n, j, s, t)$ over all j, s, t.
Complexity: $O(n^2 P_n d)$.

Justification. If $j \neq 1$ or $s \neq p_i$, the i-th job (which is longer than those already scheduled) may be placed before the other early jobs (by LPT), where it is early by $t - p_i$, or the job may be placed first (by LPT) in the last tardy batch (by Lemma 2), where it becomes tardy by $P_i - t$, increases the tardiness of the $j-1$ other jobs in the batch by p_i, and is early itself by $s - p_i$. If $j = 1$ and $s = p_i$, the job may be placed early, or it may form the last tardy batch (by Lemma 2), where it is still tardy by $P_i - t$ and incurs a new delivery cost. The complexity of the algorithm is proportional to the number of function values to be computed, since each computation requires a constant effort. The value of $c^*(i-1, t)$ was computed as the algorithm progressed through the previous stage and thus does not add to the complexity. Some effort may be saved by being more sophisticated with which $c(i, j, s, t)$ are computed (since some are obviously infeasible), but the upper bound on the complexity is $O(n^2 P_n d)$.

Algorithm 1b. Renumber the jobs such that $p_i \ge p_{i+1}$ for all i. Let $P_i = p_1 + p_2 + \cdots + p_i$. Let $f(i, j, s, t)$ be the minimum cost of scheduling the i longest jobs with j jobs in the first tardy batch where those jobs have a combined processing time of s and the combined processing times of the early jobs is t. (All jobs pushed to the beginning or end. See Figure 4.) $i = 0, \ldots, n$. $j = 0, \ldots, n$. $s = 0, \ldots, P_n$. $t = 0, \ldots, d$.

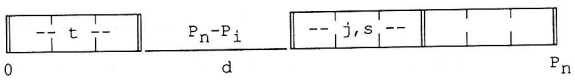

Figure 4

Initial values: $f(0, 0, 0, 0) = 0$.
$f(0, j, s, t) =$ infinity otherwise.
$f(i, j, s, t) =$ infinity if $j < 0$, $s < 0$, or $t < 0$.
Iteration:
if $j \neq 1$ or $s \neq p_i$,

$$f(i, j, s, t) = \min \begin{cases} f(i-1, j, s, t-p_i) + (d-t)\alpha, \\ f(i-1, j-1, s-p_i, t) + (P_n - P_i + t + s - d)\beta + (j-1)p_i\alpha, \end{cases}$$

if $j = 1$ and $s = p_i$,

$$f(i, j, s, t) = \min \begin{cases} f(i-1, j, s, t-p_i) + (d-t)\alpha, \\ f^*(i-1, t) + (P_n - P_i + t + s - d)\beta + K, \end{cases}$$

where $f^*(i-1, t) = \min f(i-1, j, s, t)$ for all $j, s\}$.
Update $f^*(i, t)$ if necessary.
Answer: minimum $f(n, j, s, t)$ over all j, s, t.
Complexity: $O(n^2 P_n d)$.

Justification. If $j \neq 1$ or $s \neq p_i$, the i-th job (which is smaller than those already placed) may (by Lemma 2) be placed early or be inserted into the first tardy set. If early, the job goes after (by LPT) the other early jobs, where it is early by $d - t$. In the first tardy set, it is last (by LPT), where it is tardy by $P_n - P_i + t + s - d$ and it increases the earliness of the $j - 1$ other jobs by p_i. If $j = 1$ and $s = p_i$, the job may be placed early, or it may form a tardy batch, where it is tardy by the amount $P_n - P_i + t + s - d$ and adds a delivery cost. The complexity analysis is the same as that for Algorithm 1a.

The final answer to the problem is the minimum of the answers to Algorithms 1a and 1b.

4. Polynomially-solvable special cases

For some special cases, additional assumptions lead to problems that may be solved in strictly polynomial time. One such assumption is that all jobs have the same processing time. Another is that the unit earliness penalty is zero.

4.1. A special case where all of the processing times are equal

Suppose that all of the jobs have the same processing time, that is, $p_j = p$ for all j. This case might occur in a single-item facility. Because all of the jobs are the same, only the timing of the deliveries and the starting time of the first job need to be determined. From Property III, we know that either the first job will start at time zero or that a job will finish at the due date d.

In Algorithm 2a, we consider schedules where a job finishes at the due date d. We first calculate the best way to deliver the tardy jobs and then add the cost of completing jobs early.

Define k as $[d/p]$, where $[x]$ is the greatest integer less than or equal to x. The quantity k is the maximum number of jobs that could finish before the due date. Likewise, $n - k$ is the minimum number of jobs that must be tardy.

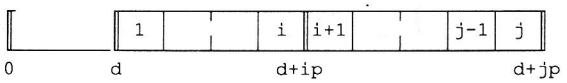

Figure 5

Algorithm 2a. Let $f(j)$ be the minimum cost incurred by j tardy jobs. Let $f(0)=0$ and define the triangular number $T_m = 1 + \cdots + m$. For $j = 1, \ldots, n$, compute $f(j)$ as follows:

$$f(j) = \min\{f(i) + (j-i)jp\beta + T_{j-i-1}p\alpha + K : i = 0, \ldots, j-1\}.$$

Then, let $h(j)$ be the minimum cost of scheduling all n jobs if j jobs are delivered tardy. Let $h(n) = f(n)$ and $h(n-1) = f(n-1)$. For $n-k \le j \le n-2$, compute $h(j)$:

$$h(j) = f(j) + T_{n-j-1}p\alpha.$$

The minimum cost to schedule all n jobs, with one ending at the due date d, is $h_{2a} = \min\{h(j) : j = n-k, \ldots, n\}$.

Justification. The formulation of $f(j)$ is similar to the dynamic lot sizing model. If i jobs have already been delivered, $j-i$ must be delivered now. See Figure 5. These jobs are tardy by an amount jp (the tardiness of the last job in the batch), and the unit penalty is β. The earliness of the last job in the batch is 0, the next to last job has earliness p, the job before that earliness $2p$ and so on to the first job in the batch, which must wait for $j-i-1$ jobs to complete and thus has earliness $(j-i-1)p$. Thus, the total earliness is $p + \cdots + (j-i-1)p = T_{j-i-1}p$, and the unit penalty is α. Finally, there is the cost K for delivering this batch.

The computation of $h(j)$ depends upon the fact that the cost of the early jobs is independent of how the tardy jobs are delivered. Thus, if j jobs are delivered tardy, $n-j$ jobs form a batch that is delivered on time. As before, the total earliness for this batch is $p + \cdots + (n-j-1)p = T_{n-j-1}p$, with a unit earliness penalty of α.

We now move to the case of starting the first job at time zero.

Algorithm 2b. Let h_{2b} be the minimum cost of a schedule that starts at time zero. Let $\delta = d - kp$. Compute h_{2b} as follows:

$$h_{2b} = h(n-k) - (n-k)\delta\beta + k\delta\alpha.$$

Justification. Note that by the definition of k, a schedule that starts at zero will have k early jobs and $n-k$ tardy jobs. If we look at a schedule with $n-k$ tardy jobs that has a job ending at the due date d, and $\delta = d - kp$, δ is the starting time of the first job. See Figure 6. By the definition of k, $kp \le d < (k+1)p$, implying $\delta < p$, so shifting a schedule to the left by δ does not make any tardy job early. Thus, there is no change in the delivery schedule. Moreover, the change in cost does not depend upon the delivery schedule. The tardy jobs are now less tardy by δ, and the early jobs are now more early by δ. Thus, the change in cost is the reduced tardiness penalties of the $n-k$ tardy jobs and the increased earliness penalties of k jobs. From this, we can deduce that the optimal scheme developed in Algorithm 2a for $n-k$ tardy jobs will yield an optimal shifted schedule. Thus, the minimum cost of starting at time zero is the minimum cost of a schedule with $n-k$ tardy jobs, that is $h(n-k)$, plus this fixed change $k\delta\alpha - (n-k)\delta\beta$.

The minimum cost for this problem is the minimum of h_{2a} and h_{2b}.

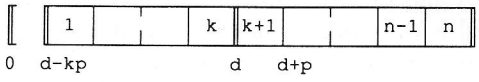

Figure 6

Complexity. If all T_j are calculated beforehand, which takes $O(n)$ time, the complexity of Algorithm 2a is $O(n)$ for each computation of $f(j)$, and the computation of $h(j)$ is constant. Thus, the total complexity is $O(n^2)$. Algorithm 2b has constant effort.

4.2. A special case with no earliness penalty

We now take up the problem where the earliness penalty is zero, that is, $\alpha = 0$. We know that after the first job starts, there is no inserted idle time. We can prove that there exists an optimal schedule that starts at time zero under the SPT ordering. Starting at zero is optimal because starting jobs earlier decreases any tardiness and, with the earliness penalty equal to zero, does not change any earliness costs. By using SPT, we finish as many jobs as possible early; the use of LPT for the early jobs is not important because there is no earliness penalty. Likewise, the LPT order in tardy batches is not necessary, although we keep the batch-SPT property that we had before.

Because the sequence of the jobs is already determined, only the timing of the deliveries remains to be found. From the above reasoning, we know that the first job will start at time zero. Thus, to solve the problem, we must calculate the best way to deliver the tardy jobs. This will done with a dynamic program similar to that in Section 5.1.

Theorem 2. *There exists an optimal schedule for this problem where the jobs are ordered by* SPT.

Proof. Suppose there exists an optimal schedule such that there exists J_i followed by J_j where $p_i > p_j$. Interchange these two jobs. Note that the completion times of all other jobs are unaffected by this, and J_i now finishes when J_j did. If we can prove that, in all cases, interchanging J_i and J_j does not increase the cost of the schedule, then repeated interchanges will yield an SPT schedule whose objective function value is no worse and is thus optimal. These two jobs must fall into one of the following cases:

A. Both jobs are early. The tardiness cost is zero. After interchanging the jobs, both still finish before the common due date, and the tardiness is still zero.
B. Both jobs start after the due date and are in the same tardy batch. Each job had the same fixed tardiness cost, which depended upon the completion time of the last job in the batch, which was unchanged by the interchange. Thus, the new tardiness costs are the same, and the change in costs is zero.
C. Both jobs start after the due date and are in different tardy batches. Thus, J_i was the last job in one batch, and J_j the first in the next. After the interchange, J_i assumes the tardiness of J_j. J_j now finishes earlier than J_i did, and the delivery time of the each job in the first batch is reduced by $p_i - p_j$. Thus, the tardiness costs decrease, with no change in the number of deliveries.
D. J_i finishes early (on or before the due date) and J_j finishes tardy. After the interchange, J_j finishes before J_i did and is thus early, while J_i finishes when J_j did and is delivered when J_j was, thus incurring the tardiness of J_j. Therefore, there is no change in the costs.
E. J_i starts before the due date and finishes after. Subcase 1: after the interchange, J_j now finishes early. While J_i inherits the finish time, delivery date, and thus tardiness costs of J_j, the tardiness of J_i is erased. Thus, the costs decrease. Subcase 2: after the interchange, J_j is tardy. This case is the same as case B or case C, depending on whether the jobs were in the same batch before.

Therefore, in all cases, the cost is not increased by the interchange. □

Now, we can sequence the jobs by SPT and renumber. Since the first job starts at time zero, the completion time of J_k is $p_1 + \cdots + p_k$, for $k = 1, \ldots, n$. Hence the lateness $F(k) = p_1 + \cdots + p_k - d$. Let i be the minimum k such that $F(k) > 0$. Thus J_i is the first tardy job. We will now use a dynamic program to find the minimum cost of delivering the tardy jobs.

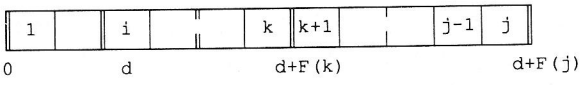

Figure 7

Algorithm 3. For $j = i, \ldots, n$, define $f(j)$ as the minimum cost incurred by the first j jobs, with the last delivery at J_j. Define $f(i-1) = 0$. For $j = i, \ldots, n$, compute $f(j)$ as follows:

$$f(j) = \min\{f(k) + (j-k)F(j)\beta + K : k = i-1, \ldots, j-1\}.$$

The answer to the problem is $f(n)$.

Justification. The formulation of $f(j)$ is similar to the dynamic lot sizing model. If k jobs have already been delivered, $j-k$ jobs must be delivered now. See Figure 7. These jobs are tardy by an amount $F(j)$ (the lateness of the last job in the batch, J_j), and the unit penalty is β. There exists no earliness costs. Finally, there is the cost K for delivering this batch.

Complexity. The sorting takes $O(n \log n)$ time; the $F(j)$ are calculated in $O(n)$ time; finally, there are $O(n)$ steps in the algorithm, and the complexity is $O(n)$ for each computation of $f(j)$. Thus, the total complexity of Algorithm 3 is $O(n^2)$.

5. An improved method for determining tardy deliveries

Given that the schedule of tardy jobs is fixed by some dominance property, as in our special cases, it remains only to determine the times of the tardy deliveries. Due to the nature of the delivery cost, it is known that these deliveries should occur only at the completion time of some tardy job and that all jobs completed and undelivered should be delivered then. It is possible to formulate a simple dynamic program to solve this problem in $O(n^2)$ time as we did in the above algorithms. However, we show here that this problem can also be transformed into a special dynamic lot sizing problem that is solvable in linear time.

The N-period dynamic lot size model can be formulated as follows (see, for example, Wagelmans, Van Hoesel and Kolen, 1992, and Chen and Lee, 1991):

$$\begin{aligned} \text{Min} \quad & \sum (c_t x_t + k_t y_t) \\ \text{s.t.} \quad & x_1 + \cdots + x_t \geq D_t \quad \text{for all } t < N, \\ & x_1 + \cdots + x_N = D_N, \\ & 0 \leq x_t \leq M y_t \quad \text{for all } t, \end{aligned}$$

where, for period t, x_t is the production, c_t is the modified unit production cost, k_t is the setup cost, D_t is the cumulative demand, $y_t = 1$ if there is production in the period and 0 otherwise, and M is some large number.

Our tardy delivery problem for the N tardy jobs (we can ignore the early jobs at this point) with earliness and tardiness penalties and common due date d is as follows:

$$\text{Min} \quad \sum (\alpha E_i + \beta T_i + K y_i)$$

where $y_i = 1$ if $D_i = C_i$ and 0 otherwise.

Note that C_i, the completion time of J_i, is fixed and that the delivery time is $D_i = d + T_i$. Thus, we can substitute $E_i = D_i - C_i = d + T_i - C_i$ into the problem:

$$\text{Min} \quad \sum ((\beta + \alpha) T_i + K y_i + \alpha(d - C_i))$$

where $y_i = 1$ if $D_i = C_i$ and 0 otherwise.

Now, define x_i as the number of jobs delivered when J_i completes, and let T_i^* be the tardiness of job J_i if it is delivered at its completion time, a known quantity. The tardiness of a job is the tardiness of its delivery date, which is the completion time of some later job. The total tardiness of the jobs delivered when J_i completes is given by $x_i T_i^*$. Thus, we can rewrite our problem:

$$\begin{aligned}
\text{Min} \quad & \sum((\beta+\alpha)x_i T_i^* + Ky_i + \alpha(d - C_i)) \\
\text{s.t.} \quad & x_1 + \cdots + x_t \le t \quad \text{for all } t, \\
& x_1 + \cdots + x_N = N, \\
& 0 \le x_i \le N y_i, \\
& y_i = 0 \text{ or } 1.
\end{aligned}$$

The first constraint is the limitation on the number of jobs available to be delivered at any completion time, $x_t \le t - x_1 - \cdots - x_{t-1}$. Define $c_i = (\beta+\alpha)T_i^*$ and note that these increase as i increases. Then, drop the $\sum\alpha(d - C_i)$ (a constant).

Now, in order to complete the transformation into the dynamic lot size problem, renumber the c_i, x_i and y_i with $j = N + 1 - i$ and define the cumulative demand $D_j = j$. This yields

$$\begin{aligned}
\text{Min} \quad & \sum(c_j x_j + Ky_j) \\
\text{s.t.} \quad & x_1 + \cdots + x_k \ge D_k \quad \text{for all } k < N, \\
& x_1 + \cdots + x_N = D_N, \\
& 0 \le x_j \le N y_j, \\
& y_j = 0 \text{ or } 1.
\end{aligned}$$

This is now exactly in the dynamic lot size model. Because of the renumbering, j is counting from the last tardy job toward the common due date. Thus, $x_1 + \cdots + x_k$ is the total number of jobs delivered during the last k tardy job completions. This total must include these last k jobs and thus is at least $D_k = k$.

Furthermore, the c_j now decrease as j increases, and due to the recent development of efficient algorithms for the dynamic lot size model, the problem can be solved in linear time. See Aggarwal and Park (1990), Chen and Lee (1991), Federgruen and Tzur (1991), and Wagelmans, Van Hoesel and Kolen (1992). Also, note that the transformation, which consists of calculating the c_j and D_j, takes linear time. Thus, the original tardy delivery problem can be solved in linear time.

Also, note that the property that no completed jobs are left undelivered after a delivery is a result of the exact requirements property of an optimal solution. That is, production is started only when the inventory has dropped to zero: $I_t x_{t+1} = 0$ for all t. $I_t = x_1 + \cdots + x_t - D_t$, where D_t is the cumulative demand to period t. In our formulation $D_j = j$. With the renumbering, I_j is the number of already-completed jobs still waiting for delivery when J_j is started. (One way to see this is to note that $I_j = N - (x_{j+1} + \cdots + x_N) - j = (N - j) - (x_{j+1} + \cdots + x_N)$, which is the number of jobs already completed minus the number of jobs already delivered.) If $x_{j+1} > 0$, i.e., if jobs were delivered after J_{j+1} (which is processed before J_j), then, in an optimal schedule, $I_j = 0$, implying no jobs can still be waiting after the delivery. Thus, that delivery must have been of all jobs available.

6. The general case

If we consider the general problem, that is, the problem with arbitrary unit earliness and unit tardiness penalties, we are confronting a more terrible monster. Although the property concerning ordering within a batch can be extended to the longest weighted processing time (LWPT) rule, the more significant result that there exists an optimal schedule where the jobs are in batch-SPT order cannot be stretched to fit this problem. This property allowed us to create a dynamic program that needed pseudo-polynomial effort to find an optimal solution. For the general problem, however, we can find no

such efficient algorithm. We would need to use a branch-and-bound scheme or some exponentially difficult dynamic program to find an optimal solution.

In this section, we will state the extension of Lemma 1 and present two counter-examples that contradict the extension of Lemma 2 we would like to have. We use the term weight to refer to a unit earliness or tardiness penalty.

Lemma 3. *In an optimal schedule, the jobs within any batch are ordered by the longest weighted processing time* (LWPT) *rule. That is,* $p_i/\alpha_i \geq p_j/\alpha_j$ *for all* J_j *that are in the same batch as and follow* J_i.

Proof. A simple interchange argument shows that if a schedule has a batch in which jobs are not in LWPT order, it cannot be optimal. □

This lemma allows us to optimally order the jobs in each batch. This includes the set of jobs that are delivered on-time and each batch of tardy jobs. Thus, we can find the optimal solution given a set of early jobs and the different tardy batches. The question of partitioning the tardy jobs into batches is the problem that will ruin our hopes of solving this problem efficiently, that is, in pseudo-polynomial time.

The earliness-tardiness problem with arbitrary unit earliness and tardiness penalties is a difficult question even without delivery costs. While known to be an NP-hard problem, it is still an open question whether the general earliness-tardiness problem is strongly NP-hard. The most general problem that can be solved in pseudo-polynomial time is the case of *agreeable ratios* (Lee et al., 1991). The jobs to be scheduled have agreeable ratios if $p_i/\alpha_i < p_j/\alpha_j$ implies that $p_i/\beta_i \leq p_j/\beta_j$ for all jobs J_i, J_j.

When considering the problem with delivery costs, equal unit earliness penalties, and equal unit tardiness penalties, we were able to order the jobs by SPT and consider the jobs one at a time, based on that list. In the more general problem, we have two weighted processing time lists: one using the earliness weights and one using the tardiness weights. If we have agreeable ratios, then these two lists can be used to form one ordering. That is, there exists an ordering of the jobs such that the quantities p_i/α_i and p_i/β_i are non-decreasing as we consider the jobs in order: $p_1/\alpha_1 \leq \cdots \leq p_n/\alpha_n$ and $p_1/\beta_1 \leq \cdots \leq p_n/\beta_n$. Thus, we have something equivalent to the single SPT ordering we used in the restricted problem. At this point we will assume that the jobs have agreeable ratios. Our problem with delivery costs is still too complex, however. The primary issue is whether the following property holds:

Property 1. If the jobs to be scheduled have agreeable ratios, then there exists an optimal schedule such that for all consecutive tardy batches S_k and S_{k+1}, if J_i is in S_k and J_j is in S_{k+1}, then $p_i/\alpha_i \leq p_j/\alpha_j$ and $p_i/\beta_i \leq p_j/\beta_j$.

If this property holds, then we may consider the jobs in the ordering given by the agreeable ratios and use a dynamic program to find an optimal solution in a manner similar to that of the restricted problem. Property 1 limits the partitioning of tardy jobs we need to do to find an optimal schedule, while Lemma 3 orders the jobs in each batch.

However, Property 1 does not necessarily hold in problems with agreeable ratios. Moreover, it is not necessarily true even if $\alpha_j = \beta_j$ for all J_j. It also may not hold if $\beta_j = \beta$ for all J_j. The examples we present show that the optimal scheduling of a set of tardy jobs does not necessarily follow Property 1. The identification and scheduling of the early jobs is not presented, and the tardy jobs are numbered in the order imposed by the agreeable ratios. Also, without loss of generality, it is assumed that the first tardy job begins at the common due date.

Example (set of tardy jobs): $K = 20$.

j	p_j	α_j	β_j
1	1	5	5
2	10	6	6
3	2	1	1

In this example, we have $\alpha_j = \beta_j$ for all J_j. The only optimal scheduling of the tardy jobs, with a batch delivery cost of 20, is $[J_3 J_1; J_2]$, where J_3 and J_1 form the first tardy batch and J_2 the second. The total cost of earliness, tardiness, and delivery is 137. Note that J_3 is in an earlier batch than that of J_2 but that J_2 has a shorter weighted processing time than J_3 $(10/6 < 2/1)$. This statement proves that Property 1 is not true. This conclusion is supported by the next example also.

Example (set of tardy jobs): $K = 20$.

j	p_j	α_j	β_j
1	1	2	5
2	2	2	5
3	2	1	5

In this example, we have $\beta = \beta_j$ for all J_j. The only optimal scheduling of the tardy jobs, with a batch delivery cost of 20, is again $[J_3 J_1; J_2]$, where J_3 and J_1 form the first tardy batch and J_2 the second. The total cost of earliness, tardiness, and delivery is 96. Again, J_2 has a shorter weighted processing time than J_3 $(2/2 < 2/1)$ and is in a later batch than that of J_3.

So, Property 1 is not true. While is it possible to write a dynamic program to solve this problem, the algorithm would have to include state variables for every possible tardy batch, and the maximum number of tardy batches is equal to the total number of jobs. Thus, the algorithm would need an exponential amount of space and effort. Due the difficulty of finding exact solutions to the general problem, we have concentrated in this paper on the cases we can solve with polynomial or pseudo-polynomial effort. We leave the general problem and its arbitrary unit earliness and tardiness penalties for future research.

7. Empirical results

In addition to examining the theoretical complexity of the dynamic programming algorithm, we have experimented with a Pascal implementation of the algorithm. We were able to solve problems with up to 50 jobs in about three minutes of CPU time, and the amount of computer processing time needed to solve a problem was proportional to a polynomial function of the number of jobs, the due date, and the total processing time. Due to the large amount of time necessary to process larger problems, we later developed and tested a number of simple heuristics, described in Section 8.

In order to estimate how efficiently dynamic programming solved this problem, we ran a Pascal implementation of the algorithm under the VMS operating system on a Vax 6320. The goal of these tests was to sample the computer processing time required to solve different-sized problems and to compare the actual computational effort to the theoretical $O(n^2 P_n d)$ limit on the effort.

With the dynamic program we solved 17 sample problems that ranged in size from 10 to 50 jobs. We created these sample problems by hand with varying due dates, processing times, and other job characteristics. We solved each problem by finding the answer to both dynamic programming algorithms (one around the due date and one starting at time zero). The computational effort required was measured and plotted versus the theoretical effort. This showed that the computational effort was not proportional to the quantity $n^2 P_n d$. The problems with larger d required less effort than expected. This, as we will now show, is due to the details of the dynamic programming iteration.

On a given step of the implemented version of Algorithm 1a (or Algorithm 1b), the i-th job is being added. The loops in this iteration are for $j = 0, \ldots, i$, $t = 0, \ldots, \min\{d, P_i\}$ and $s = 0, \ldots, P_i - t$. Focusing on the loops over s and t, we can see that if d is greater than or equal to P_i, the effort on these loops is $\frac{1}{2}P_i^2$. If d is small, t is bounded above by d and s can range to nearly P_i. Thus, the effort is close to $P_i d$, which was used in computing the theoretical limit.

Table 1
Computational time versus theoretical effort

Problem No.	Jobs	Due date	Makespan	CPU time	Effort $n^2(P_n d - \frac{1}{2}d^2)$
1	10	20	55	3.0	90000
2	10	20	55	3.0	90000
3	20	10	40	5.0	140000
4	20	20	40	8.1	240000
5	20	40	40	9.0	320000
6	20	40	60	18.3	640000
7	20	40	100	39.5	1280000
8	20	40	100	40.0	1280000
9	20	40	108	45.0	1408000
10	20	84	84	40.0	1411200
11	20	84	84	40.3	1411200
12	30	70	70	57.2	2205000
13	30	70	70	52.8	2205000
14	30	50	100	103.0	3375000
15	40	30	100	127.7	4080000
16	40	50	100	176.2	6000000
17	50	30	100	199.9	6375000

However, a little bit more analysis reveals that the effort over t and s is always $P_i \min\{d, P_i\} - \frac{1}{2}\min\{P_i, d\}^2$. Using the last iteration as an upper bound for the rest, with $\min\{d, P_n\} = d$, since a problem with $d > P_n$ can be solved by letting $d = P_n$ and then shifting the optimal schedule to meet the original due date, the effort of the implemented algorithm should be $O[n^2(P_n d - \frac{1}{2}d^2)]$. Plotting the processing time versus the new effort shows that the processing time is indeed nearly proportional to this quantity.

The results of this sampling can be found in Table 1 and Figure 8. Table 1 lists the problems that were studied, the problem sizes, the CPU time required to solve each problem in seconds, and the $n^2(P_n d - \frac{1}{2}d^2)$ effort. Figure 8 plots the CPU time in seconds versus the effort in thousands, clearly showing the correlation over these sample problems.

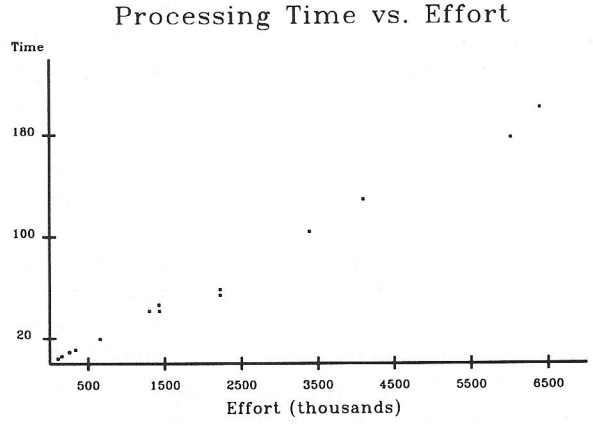

Figure 8. Plot of CPU time in seconds versus effort in thousands

8. Heuristics

It is often useful to have a number of heuristics that can be used to find good solutions to a complex problem. Heuristics differ from exact methods in that they do not always find an optimal solution and they are often simpler, appealing to intuitive or basic properties. The quality of a heuristic varies, depending on the instance or class of instances that need to be solved. Still, if it performs well on average over a range problems, the heuristic can be a useful tool for finding good solutions quickly.

We can solve our earliness–tardiness problem (if all $\alpha_j = \alpha$ and $\beta_j = \beta$) with the exact dynamic programming procedure described above. Due to the pseudo-polynomial nature of its complexity, however, the algorithm may require more computational effort or memory than the person wishing to solve the problem has. Thus, we developed a class of heuristics that are intuitively simple and tried measure how well they perform over a range of different problems. As we will see, the ability of the heuristics to find good solutions varies over the different classes of problem instances.

One intuitive approach to the earliness–tardiness problem we are studying is to divide the jobs to be scheduled into two sets, those to be early and those to be tardy, and schedule each set optimally.

Each heuristic performs the second step by scheduling the early jobs by LPT to end at the common due date and by using a quadratic dynamic program to determine the deliveries of the tardy jobs, starting at the common due date. Additionally, if there is a gap before the start of the first early job, each heuristic shifts the schedule to the left to create a schedule that starts at time zero. (This can be done only if no tardy job is made early by the shift.) This shift may improve the quality of the solution by some small amount, especially if many jobs are tardy or the unit tardiness penalty is large.

The key to finding the optimal solution to a common due date earliness–tardiness problem is the division of the jobs into the early and tardy sets. If there are no delivery costs and the due date is large enough, it is optimal to divide the jobs based on positional weights. The largest jobs are scheduled as the most early and most tardy, and the smallest jobs go around the due date. With a restrictive due date, however, the number of early jobs and their sizes are constrained (since a schedule must start no earlier than time zero). Moreover, in the presence of delivery costs, the positional weights for tardy jobs cannot be determined beforehand.

In dividing the jobs into two sets, the heuristics we developed ignore the batch deliveries and the associated increases in earliness and tardiness penalties. Only one of the heuristics is explicitly concerned with the weights (the unit earliness and tardiness penalties). All, however, are concerned with the restrictiveness of the due date, since that date is crucial in creating sets that will yield a feasible schedule. If too many jobs are placed into the set of early jobs, no feasible schedule can be created.

Split 1: Order the jobs by LPT. Create positional weights, and allocate jobs to these weights by matching the longest unscheduled job to smallest unused weight. If a job will not fit into the early set, make it tardy.

Split 2: Order by SPT. Starting with the early set, allocate jobs by alternating between the two sets. If a job will not fit into the early set, make it and all subsequent jobs tardy.

Split 3: Order by LPT. Starting with the early set, allocate jobs by alternating between the two sets. If a job will not fit into the early set, place it into the tardy set. (The opposite of Split 2, this is also a simpler version of Split 1.)

Split 4: Order by LPT. Starting with the early set, allocate jobs by alternating between the two sets. If a job will not fit into the early set, make it tardy. In addition, if the tardy set is too large (see below), place the job in the early set. (An extension of Split 3.)

Split 5: If it will fit, place the largest job into the early set. Order the remaining jobs by SPT, and allocate jobs to the early set until no more will fit. The set of unscheduled jobs becomes the tardy set.

Table 2
Problem sets created for testing

Set	Jobs	Times [a]	α	β	K	$d/\Sigma P_j$
ETA15	15	[1, 10]	1	2	20	$\frac{1}{2}$
ETB15	15	[1, 30]	1	2	20	$\frac{3}{4}$
ETC15	15	[10, 20]	1	1	5	$\frac{3}{4}$
ETD15	15	[1, 20]	1	2	100	$\frac{1}{2}$
ETE15	15	[1, 20]	1	2	200	$\frac{1}{2}$
ETA30	30	[1, 20]	1	2	100	$\frac{1}{2}$

[a] Processing times sampled from a uniform distribution with the given range.

As mentioned before, the schedule created around the due date can be shifted to start at time zero if this shift makes no tardy job on-time. With a large common due date, some of the splits will create an early set that does not fill the space between zero and the due date, implying that the tardy set is large. When the schedule is shifted, one or more of the excessive jobs in the tardy set may become early. Splits 4 and 5 alleviate this problem in different ways. Split 4 explicitly limits the size of the tardy set to ensure that no job could complete before the due date when the schedule is shifted. That is, if the current size of the tardy set is larger than or equal to the difference between the total processing time and the common due date, no jobs can be added to the set.

Split 5 limits the size of the tardy set by filling the early set to capacity first. While Split 1 works from what positional weights can be assigned, Split 5 simply places the largest job early, where it contributes to no job's earliness or tardiness but instead reduces the number of jobs that can be made early. The split then compensates for that by placing as many jobs early as possible.

In order to test the effectiveness of the heuristics, we created six sets of random problems, with five problems in each set. The due date was set at a fraction of the sum of the processing times. Each processing time was chosen from a uniform distribution, while the other problem characteristics were given pre-selected values. The optimal solutions were found using the dynamic program. While two different schedules (the one around the due date and the one shifted to start at time zero) could be created by a heuristic, the best of these two was selected as the output of the heuristic.

The performance of the heuristics is mixed over the problem sets. However, the average performance of Split 5 is usually good. On the second problem set, both Split 1 and Split 4 perform well. In this set the problems have large common due dates, implying that there is plenty of room to place early jobs, which are desirable due to the larger tardiness penalty. On the third problem set, the two simple splits work well since the unit earliness and tardiness penalties are equal, the due date is large, and the delivery cost is small, implying that very little batching occurs. Also, we note that each heuristic runs very quickly, since it only needs to sort the jobs, which takes $O(n \log n)$ effort, and then consider the jobs in that order.

Table 3
Performance of heuristics on problem sets [a]

Set	Jobs	Split 1	Split 2	Split 4	Split 5
ETA15	15	24.2	19.1	13.3	4.9
ETB15	15	2.5	22.5	1.8	4.3
ETC15	15	0.0	0.0	13.7	19.3
ETD15	15	24.3	22.9	13.8	6.2
ETE15	15	26.7	18.0	16.9	8.7
ETA30	30	21.4	6.4	9.1	4.5

[a] Performance is mean deviation from optimal on five problems.

9. Concluding notes

In this paper we have studied a scheduling problem that extends the earliness–tardiness problems previously investigated. We considered a delivery cost for each batch of tardy jobs, a cost which adds a considerable amount of difficulty to even the simplest earliness–tardiness problems. We spent most of our time in this paper looking at the equally-weighted case, where all of the unit earliness penalties are equal and all of the unit tardiness penalties are equal. We were able to solve this problem with a pseudo-polynomial dynamic program, and we were able to implement it to solve problems with up to 50 jobs. We investigated a number of simple heuristics that can find good solutions to different classes of instances for the problem. Additionally, we considered both a pair of special cases that could be solved in polynomial time and the general problem (where the jobs have different weights) for which no efficient algorithm could be found.

We have assumed that the common due date d is given and restrictive. It is an open question whether the problem is NP-complete if d is unrestrictive in the standard sense or if d is a decision variable. If the batch delivery cost K is zero, the problem is polynomial because positional weights can be assigned and the problem solved by matching the largest weights with the shortest jobs. However, with the batch-flow problem, the positional weights depend upon the delivery schedule, which cannot be determined without the job information. If d is indeed a decision variable, at worst the problem can be solved in pseudo-polynomial time by letting $d = P_n$ and using Algorithm 1a to find the optimal solutions. Then, d can be set to a value of t that is in an optimal solution.

This paper leaves a number of areas open for further research. More analysis of the general problem is necessary in order to determine properties or cases that could lead to efficient solution techniques or good heuristic solutions. Also, as in Liman and Lee (1992), work could be done on developing error bounds for the heuristics that we did develop for our problem.

Acknowledgements

This material is based upon work supported under a National Science Foundation Graduate Fellowship. Part of this research was undertaken while the second author was a visiting scholar at the Department of Actuarial and Management Sciences, University of Manitoba, Canada. This author is thankful to Professor T.C.E. Cheng for his helpful comments and invitation to visit the University of Manitoba. We would also like to thank the Editor, Dr. Jatinder N.D. Gupta, and two anonymous referees for their helpful comments.

Bibliography

Aggarwal, A., and Park, J.K. (1990), "Improved algorithms for economic lot-size problem", Working Paper, IBM Thomas J. Watson Research Center, Yorktown Heights, New York.

Baker, K.R., and Scudder, G.D. (1990), "Sequencing with earliness and tardiness penalties: A review", *Operations Research* 38/1, 22–36.

Chen, H.D., and Lee, C.-Y. (1991), "A simple algorithm for the error bound of the dynamic lot size model allowing speculative motive", Research Report 91-5, Dept. of Industrial and Systems Engineering, University of Florida, Gainesville, FL.

Cheng, T.C.E., and Kahlbacher, H.G. (1991), "Scheduling with delivery and earliness penalties", Working paper, Dept. of Actuarial and Management Sciences, University of Manitoba, Canada.

Dobson, G., Karmarkar, U.S., and Rummel, J.L. (1987), "Batching to minimize flow times on one machine", *Management Science* 33/6, 784–799.

Federgruen, A., Mosheiov, G. (1991), "Efficient algorithms for scheduling problems with general earliness and tardiness cost structures", Graduate School of Business, Columbia University.

Federgruen, A., and Tzur, M. (1991), "A simple forward algorithm to solve general dynamic lot sizing models with n period in $O(n \log n)$ or $O(n)$ time", *Management Science* 37/8, 909–925.

Hall, N.G., and Posner, M.E. (1991), "Earliness–tardiness scheduling problems, I: Weighted deviation of completion times about a common due date", *Operations Research* 39/5, 836–846.

Hall, N.G., Kubiak, W., and Sethi, S.P. (1991), "Earliness–tardiness scheduling problems, II: Deviation of completion times about a restrictive common due date", *Operations Research* 39/5, 847–856.

Lee, C.-Y., Danusaputro, S.L., and Lin, C.S. (1991), "Minimizing weighted number of tardy jobs and weighted earliness–tardiness penalties about a common due date", *Computers & Operations Research* 18, 379–389.

Liman, S.D., and Lee, C.-Y. (1992), "Error bound for a heuristic on the common due-date scheduling problem", accepted by *ORSA Journal of Computing*.

Wagelmans, A., Van Hoesel, S., and Kolen, A. (1992), "Economic lot-sizing: an O(n log n) algorithm that runs in linear time in the Wagner–Whitin case", *Operations Research* 40, S145–S156.