INFORMS Journal on Computing
Vol. 9, No 1, Winter 1997

0899-1499/97/0901-0092 $05.00
© 1997 INFORMS

Earliness–Tardiness Scheduling Around Almost Equal Due Dates

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

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

(Received: May 1993; revised: February and August 1995; accepted: February 1996)

The just-in-time concept in manufacturing has aroused interest in machine scheduling problems with earliness–tardiness penalties. In particular, common due date problems, which are structurally less complicated than problems with general due dates, have emerged as an interesting and fruitful field of research. We prove that so-called almost common due date problems, in which the due date d_j and processing time p_j of each job J_j ($j = 1, \dots, n$) are such that $d_j \in [D, D + p_j]$ for some constant D, are structurally less complicated also. Our main contribution is an $O(n^2)$ time dynamic programming algorithm for the almost common due date problem with large D. The dynamic programming algorithm is interesting in its own right, since the optimality principle behind it applies to other common due date and almost common due date problems as well.

The interest in the just-in-time concept in manufacturing has prompted research on machine scheduling problems with earliness and tardiness penalties; for a survey, see [1]. The problem in the focus of attention is the following. There are n jobs J_j ($j = 1, \dots, n$) that have to be processed by a single machine that is continuously available from time zero onward and that can handle no more than one job at a time. Each J_j requires processing during an uninterrupted period of given length p_j and ideally should be completed at its due date d_j. Without loss of generality, we assume that p_j and d_j are integral. A *schedule* σ specifies for each job J_j a completion time $C_j(\sigma)$ such that no two jobs overlap in their execution; we simply write C_j if there is no confusion possible as to the schedule we are referring to. Given any σ, the *earliness* and *tardiness* of J_j are defined by $E_j = \max\{0, d_j - C_j\}$ and $T_j = \max\{0, C_j - d_j\}$; correspondingly, a job is called *early* if it is completed before its due date and *tardy* if it is completed after its due date. If a job is completed exactly at its due date, then it is called *just-in-time*. The quality of σ is measured by the objective function $f(\sigma) = \sum_{j=1}^{n} [\alpha_j E_j + \beta_j T_j]$, where the weights α_j and β_j are given positive integers. This type of objective function, penalizing both early and tardy completions, reflects the premise of just-in-time manufacturing. The problem is to find a schedule with minimum objective value.

The general problem is NP-hard in the strong sense, since the special case $\alpha_j = 0$ for each j reduces to the total weighted tardiness problem, which is NP-hard in the strong sense.[9,11] The problem is also very hard from a computational point of view, even if all α_j and β_j are replaced with a common α and β, as reported in [7]. The special case in which all due dates are equal, that is, $d_j = d$ for all j, has received much attention. It is known as the *common due date problem*, and it splits up into two variants, depending on whether d is *large*, i.e., $d \geqslant \sum_{j=1}^{n} p_j$, or *small*.

For d large, the problem with unit weights and the problem with $\alpha_j = \alpha$ and $\beta_j = \beta$ for all j are solvable in $O(n \log n)$ time.[2,8] For distinct symmetric weights, that is $\alpha_j = \beta_j$ for all j, the problem is NP-hard but solvable in $O(n \sum_{j=1}^{n} p_j)$ time and space.[4] It is an open question whether the common due date problem with general weights is NP-hard in the strong sense.

For d small, the problem with unit weights is already NP-hard in the ordinary sense.[3,6] Hall, Kubiak and Sethi[3] also provide a pseudo-polynomial algorithm requiring $O(n \sum_{j=1}^{n} p_j)$ time and space. For distinct symmetric weights the problem is solvable in $O(n^2 \sum_{j=1}^{n} p_j)$ time and $O(n \sum_{j=1}^{n} p_j)$ space.[6]

In this article, we establish that, in addition to the class of common due date problems, there exists another class of problems that are structurally less complicated than the general earliness–tardiness problem. This class consists of problems in which the due dates d_j may differ but all lie in the interval $[D, D + p_j]$ for some given D. We call such due dates *almost common due dates* and problems in this class *almost common due date problems*. Note that this class encompasses common due date problems; accordingly, almost common due date problems are at least as hard as common due date problems. We show that each variant of the almost common due date problem belongs to the same complexity class as its common due date counterpart. Our main contribution is an $O(n^2)$ time dynamic programming algorithm for the large almost common due date problem with $\alpha_j = \alpha$ and $\beta_j = \beta$ for all j ($j = 1, \dots, n$). The dynamic programming algorithm is interesting in its own right, since the optimality principle behind it applies also to other large common due date and large almost common due date problems.

As an example, consider the following situation. We are

Subject classifications: Production scheduling
Other key words. sequencing, deterministic models

given n jobs that have to be executed by a bottleneck machine first, after which these half-products need to undergo the finishing touch on any nonbottleneck machine. Since the nonbottleneck machines have plenty of capacity, they are only used from say time D_1 to time D_2, whereas the bottleneck machine is continuously available. You wish to have the half-products ready at time D_1, so that the nonbottleneck machines can start processing. If the half-product is completed before time D_1, then it cannot immediately be finished, which might decrease the quality of the product. Furthermore, you want to deliver the products as soon as possible, which implies that the half-products should be finished on the nonbottleneck machine as early as possible after time D_1. This situation might be modeled as a common due date problem, where each job should ideally be completed at time D_1 by the bottleneck machine. You may need some time, however, to make the nonbottleneck machines ready for processing, for instance if some tools need to be loaded. Then the nonbottleneck machine cannot process J_j before time $D_1 + s_j$, where s_j is the time needed to make the machine ready to execute J_j. Therefore, the specific due dates should not be set equal to D_1 but to $D_1 + s_j$. If $s_j \leq p_j$, then this model falls in our class of almost common due date problems.

This article is organized as follows. In Section 1, we recall Emmons' matching algorithm. The logic behind it is employed in our $O(n^3)$ time dynamic programming algorithm for the large almost common due date problem, which we present in Section 2. In Section 3, we speed up the algorithm to run in $O(n^2)$ time. In Section 4, we show that the other variants of the almost common due date problem have the same computational complexity as their common due dates counterparts. In Section 5, we identify other earliness–tardiness problems to which the optimality principle behind our dynamic algorithm applies. Section 6 summarizes and concludes the paper.

1. Basics for the Large Common Due Date Problem

Kanet[8] presents an $O(n \log n)$ algorithm for the large common due date problem with $\alpha = \beta$. Emmons[2] proposes an $O(n \log n)$ algorithm for the more general case that $\alpha \neq \beta$. The concepts of these algorithms are prerequisites for the subsequent sections.

Theorem 1 (see [8]). *No optimal schedule has idle time between the execution of the jobs.*

In the remainder, we call a schedule with no idle time between the execution of the jobs a *compact* schedule.

Theorem 2 (see [8]). *There is an optimal schedule for which the due date d coincides with either the start or the completion time of the job with the smallest processing time.*

Emmons' algorithm is based upon the concept of positional weights and positional costs. The cost of assigning J_j to the kth early position, that is the kth position in the schedule, is equal to $\alpha(k-1)p_j$; the cost of assigning J_j to the kth tardy position, that is the $(n+1-k)$th position in the schedule, is equal to $\beta k p_j$. The scheduling problem thus reduces to a special case of *matching* in which jobs have to be assigned to positions; this problem is solved in $O(n \log n)$ time by matching the job that has the kth largest processing time with the position that has the kth smallest weight, for $k = 1, \ldots, n$. Consequently, any optimal schedule must be *V-shaped,* that is, the jobs completed before or at d must be scheduled in order of nonincreasing processing times and the jobs started at or after d must be scheduled in order of nondecreasing processing times.

2. The Large Almost Common Due Date Problem with $\alpha_k = \alpha$ and $\beta_k = \beta$

We start with the large variant of the common due date problem; that is, the case in which each due date d_j lies in the interval $[D, D + p_j]$ with D no less than the total processing time of all jobs. By analogy with the large common due date problem, there exists an optimal schedule σ^* for the large almost common due date problem without idle time between the execution of jobs, and with at least one job completed exactly at its due date; we select one of these jobs and call it the *pivotal* job or the *pivot.* Due to the due date structure, *all jobs before the pivot are early or just-in-time and all jobs after the pivot are tardy or just-in-time.* If no confusion is possible, then we simplify the wording by calling all jobs before the pivot *early* and by calling all jobs after the pivot *tardy.* Let $J_{[g]}$ denote the job that occupies the gth position in σ^*, and let $p_{[g]}$ and $d_{[g]}$ be defined correspondingly; suppose that the pivot occupies the mth position in the schedule, that is, $J_{[m]}$ is the pivot. Then we have that

$$\begin{aligned} f(\sigma^*) &= \alpha \sum_{g=1}^{m} (d_{[g]} - C_{[g]}) + \beta \sum_{m+1}^{n} (C_{[g]} - d_{[g]}) \\ &= \alpha \sum_{g=1}^{m} \left(d_{[g]} - \left(d_{[m]} - \sum_{h=g+1}^{m} p_{[h]} \right) \right) \\ &\quad + \beta \sum_{g=m+1}^{n} \left(\left(d_{[m]} + \sum_{h=m+1}^{g} p_{[h]} - d_{[g]} \right) \right) \qquad (1) \\ &= \alpha \sum_{g=1}^{m} [(g-1)p_{[g]} + d_{[g]} - d_{[m]}] \\ &\quad + \beta \sum_{g=m+1}^{n} [(n+1-g)p_{[g]} + d_{[m]} - d_{[g]}]. \end{aligned}$$

Reverting to the concept of positional costs, we see that for any given pivotal job J_q,

$$e_{jk}^{q} = \alpha[(k-1)p_j + d_j - d_q]$$

is the cost of assigning J_j $(j = 1, \ldots, n, j \neq q)$ to the kth $(k = 1, \ldots, n-1)$ position in the schedule, if this position is located to the left-hand side of the pivot, that is, $m > k$; and that

$$t_{jk}^{q} = \beta[kp_j + d_q - d_j]$$

is the cost of assigning J_j to the $(n+1-k)$th $(k = 1, \ldots, n-1)$ position in the schedule, that is, the kth position counted from behind, if this position is located to the right-hand side of the pivot, that is, $m < n + 1 - k$. For any given pivot, the cost of assigning a job to a position can thus be computed directly and independently of the other jobs. From Eq. 1 it also follows that any optimal schedule for the almost common due date problem must be V-shaped, except for the pivot, which can be any job. Hence, the search for an optimal solution over all schedules reduces to a search over all pivots and all sets of early jobs.

Reindex the jobs according to nonincreasing processing times, settling ties according to nondecreasing due dates. For any pivotal job J_q, let $\sigma_j^q(i)$ denote any compact V-shaped schedule for the jobs $J_1, \ldots, J_j$ with i jobs scheduled before the pivot. There are two possible ways to insert J_{j+1} $(J_{j+1} \neq J_q)$ in $\sigma_j^q(i)$ without destroying the V-shape of the schedule: either before J_q and immediately after all early jobs, or after J_q and immediately before all tardy jobs. In either case, the positional cost of the job that we have inserted depends upon the *number* of early jobs, not upon their identity. Hence, the following dominance rule applies.

Dominance Rule 1. *Let $\sigma_j^q(i)$ and $\pi_j^q(i)$ be two different compact V-shaped schedules for the jobs $J_1, \ldots, J_j$ with pivot J_q with i jobs scheduled before J_q. If $f(\sigma_j^q(i)) < f(\pi_j^q(i))$, then $\pi_j^q(i)$ can never lead to an optimal schedule and hence can be discarded. If $f(\sigma_j^q(i)) = f(\pi_j^q(i))$, then either $\pi_j^q(i)$ or $\sigma_j^q(i)$ can be discarded.*

The Dominance Rule gives rise to the following dynamic programming algorithm. For any pivotal job J_q, let $F_j^q(i)$ denote the minimum positional cost for the jobs $J_1, \ldots, J_j$ subject to the condition that i jobs have been assigned to positions before the pivot J_q; the positional cost of the pivot is added when the number of early jobs is known. The initialization is then

$$F_j^q(i) = \begin{cases} 0, & \text{if } j = 0,\ i = 0, \\ \infty, & \text{otherwise,} \end{cases}$$

and the recursion for $j = 1, \ldots, n$, and $i = 0, \ldots, j-1$ is

$$F_j^q(i) = \begin{cases} F_{j-1}^q(i), & \text{if } j = q, \\ \min\{F_{j-1}^q(i-1) + e_{ji}^q,\ F_{j-1}^q(i) + t_{j,j-i}^q\}, & \text{if } j \neq q. \end{cases}$$

Note that the positional cost of the pivot still has to be added to $F_n^q(i)$ to make it equal to the cost of the schedule. The positional cost of the pivot J_q is equal to $\alpha i p_q$, since the earliness of each of the i jobs scheduled before J_q contains the component p_q. For any given pivot J_q, the optimal solution value is then equal to $\min_{0 \le i \le n-1}(F_n^q(i) + \alpha i p_q)$, and it takes $O(n^2)$ time to compute it, since for each of the $(n-1)$ jobs that are added we have to evaluate $O(n)$ possibilities concerning the number of early jobs. Consequently, we have that

$$f(\sigma^*) = \min_{1 \le q \le n, 0 \le i \le n-1} (F_n^q(i) + \alpha i p_q),$$

from which we immediately learn which job should serve as the pivot. We then perform the dynamic programming algorithm again, with J_q as pivot, while storing the cost of all nondominated compact schedules. The corresponding optimal schedule is then found by backtracking, that is, by performing the recursion from $f(\sigma^*)$ backward in the opposite order from job J_n down to J_1 and recording the identity of the early jobs. Hence, the large almost common due date problem with $\alpha_j = \alpha$ and $\beta_j = \beta$ for all j can be solved by dynamic programming in $O(n^3)$ time and $O(n^2)$ space.

3. Speeding Up the Dynamic Programming Algorithm

In this section, we present a refinement of our dynamic programming algorithm to make it run in $O(n^2)$ time and space. The key to this speed-up is the combination of a more powerful dominance rule that needs no specification of the pivotal job, on the one hand, and a partial characterization of an optimal solution, on the other hand.

First, we need some additional notation. Let $\sigma_j^-(i)$ be a compact V-shaped schedule for the jobs $J_1, \ldots, J_j$ around an *unspecified* pivotal job belonging to the set $\{J_{j+1}, \ldots, J_n\}$ with i jobs scheduled before the pivot. We let $e_{jk}^- = \alpha[(k-1)p_j + d_j]$ be the *pivot-independent* cost of assigning J_j to the kth early position in such a schedule and $t_{jk}^- = \beta(kp_j - d_j)$ the pivot-independent cost of assigning J_j to the kth tardy position. Finally, let $g(\sigma_j^-(i))$ be the sum of the pivot-independent positional costs in $\sigma_j^-(i)$.

Let $\sigma_j^q(i)$ be the schedule obtained from $\sigma_j^-(i)$ by letting $J_q \in \{J_{j+1}, \ldots, J_n\}$ be the pivot. Then we have for any schedule $\sigma_j^-(i)$ and any $J_q \in \{J_{j+1}, \ldots, J_n\}$ that

$$f(\sigma_j^q(i)) - g(\sigma_j^-(i)) = \alpha i p_q - \alpha i d_q + \beta(j-i)d_q,$$

which is a constant for fixed i, j, and q. This observation gives rise to the following dominance rule.

Dominance Rule 2. *Let $\sigma_j^-(i)$ and $\pi_j^-(i)$ be two different compact V-shaped schedules for the jobs $J_1, \ldots, J_j$ around an unspecified pivotal job belonging to the set $\{J_{j+1}, \ldots, J_n\}$ with i jobs scheduled before the pivot. If $g(\sigma_j^-(i)) < g(\pi_j^-(i))$, then no $\pi_j^q(i)$ with $J_q \in \{J_{j+1}, \ldots, J_n\}$ can lead to an optimal schedule; hence, each $\pi_j^q(i)$ with $J_q \in \{J_{j+1}, \ldots, J_n\}$ can be discarded. If $g(\sigma_j^-(i)) = g(\pi_j^-(i))$, then either $\pi_j^q(i)$ or ${}_j^q(i)$ can be discarded, for any pivot $J_q \in \{J_{j+1}, \ldots, J_n\}$.*

Dominance Rule 2 implies that in practice a considerable speed-up can be achieved by first generating all schedules $\sigma_j^-(i)$ $(j = 1, \ldots, n-1, i = 0, \ldots, j)$ that cannot be discarded on basis of Dominance Rule 2 and by consequently expanding each schedule $\sigma_j^{j+1}(i)$ to a complete schedule by use of the dynamic programming algorithm described in Section 2. We call the schedules that cannot be discarded the *relevant* schedules. Note that the set containing all $O(n^2)$ relevant schedules $\sigma_j^-(i)$ can be obtained in $O(n^2)$ time altogether by a dynamic programming algorithm similar to the one described in Section 2.

The next theorem enables a speed-up for expanding the relevant schedules $\sigma_j^{j+1}(i)$; in this sense, it is supplementary to Dominance Rule 2. The proof, consisting of a case-by-case analysis, is lengthy and does not contribute to the understanding of the speed-up; it is therefore put in Appendix A.

Theorem 3. *There exists an optimal schedule for the almost common due date problem with some pivotal job J_q in which each* *job J_j with $j > q$ is scheduled before the pivot if and only if*

$$d_j - p_j \leq d_q - p_q,$$

or

$$d_j < d_q \quad \text{and} \quad \alpha(d_j - p_j - d_q + p_q) \leq \beta(d_q - d_j).$$

In the remainder, we let by default $\sigma_0^1(0)$ be the empty schedule and $f(\sigma_0^1(0)) = 0$. For $q = 1, \ldots, n$ and $i = 0, \ldots, q - 1$, let $\theta^q(i)$ be the complete schedule obtained from the relevant schedule $\sigma_{q-1}^q(i)$ by positioning the jobs $J_{q+1}, \ldots, J_n$ as dictated by Theorem 3. Theorem 3 guarantees that an optimal schedule σ^* can then be found as $f(\sigma^*) = \min_{1 \leq q \leq n, 0 \leq i \leq q-1} f(\theta^q(i))$.

For any pivot J_q, the expansion of the relevant schedules $\sigma_{q-1}^q(i)$ $(i = 0, \ldots, q - 1)$ can be implemented such that the costs $f(\theta^q(0)), \ldots, f(\theta^q(j - 1))$ can collectively be computed in $O(n)$ time. This is possible since the expansion of $\sigma_{q-1}^q(i)$ is identical for each i $(i = 0, \ldots, q - 1)$. Let $\mathcal{B}^q$ be the set of jobs J_j with $j > q$ that have to be scheduled before J_q, and let $\mathcal{A}^q$ be the set of jobs J_j with $j > q$ that have to be scheduled after J_q. In the remainder, we let $p(\mathcal{S}) = \Sigma_{J_j \in \mathcal{S}} p_j$ for any set of jobs $\mathcal{S}$. Let $\Delta^q(i)$ denote the cost of expanding $\sigma_{q-1}^{-}(i)$ to $\theta^q(i)$. Then we have that

$$\Delta^q(i + 1) = \Delta^q(i) + \alpha p(\mathcal{B}^q) - \beta p(\mathcal{A}^q) + \alpha(p_q - d_q) + \beta d_q, \quad \text{for each} \quad i = 0, \ldots, q - 2.$$

After all, each job in $\mathcal{B}^q$ when added to $\sigma_{q-1}^q(i + 1)$ faces an early position with weight α larger than the weight of the position it faces when added to $\sigma_{q-1}^q(i)$. Similarly, each job in $\mathcal{A}^q$ when added to $\sigma_{q-1}^q(i + 1)$ faces a tardy position with weight β smaller than the weight of the position it faces when added to $\sigma_{q-1}^q(i)$. The last two terms follow from the adjustment of the pivot-independent positional cost to positional costs for a known pivot. The implication is that, once we have computed $\Delta^q(0)$, which takes $O(n)$ time, each $f(\theta^q(i))$ $(i = 0, \ldots, q - 1)$ is computed in constant time as

$$f(\theta^q(i)) = f(\sigma_{q-1}^q(i)) + \Delta^q(0) + i(\alpha p(\mathcal{B}^q) - \beta p(\mathcal{A}^q) + \alpha(p_q - d_q) + \beta d_q).$$

Dominance Rule 2 and Theorem 3 in tandem with the analysis above lead to the following $O(n^2)$ algorithm for the almost common due date problem.

Almost Common Due Date Algorithm

Step 1. Reindex the jobs in order of nonincreasing processing times, settling ties according to nondecreasing due dates.
Step 2. Determine the relevant schedules $\sigma_j^-(i)$ $(j = 1, \ldots, n, i = 0, \ldots, j)$.
Step 3. For each J_q $(q = 1, \ldots, n)$, identify $\mathcal{A}^q$ and $\mathcal{B}^q$, and compute $p(\mathcal{A}^q)$, $p(\mathcal{B}^q)$, and $\Delta^q(0)$.
Step 4. $q \leftarrow 1$.
Step 5. Obtain the relevant schedules $\sigma_{q-1}^q(i)$ from $\sigma_{q-1}^-(i)$ $(i = 0, \ldots, q - 1)$, and compute $f(\theta^q(i)) = f(\sigma_{q-1}^q(i)) + \Delta^q(0) + i(\alpha p(\mathcal{B}^q) - \beta p(\mathcal{A}^q) + \alpha(p_q - d_q) + \beta d_q)$ for each i $(i = 0, \ldots, q - 1)$.
Step 6. $q \leftarrow q + 1$; if $q \leq n$, then go to Step 5.
Step 7. Determine the minimum cost and the corresponding schedule.

4. The Problem with Distinct Symmetrical Weights

The almost common due date problem with distinct symmetrical weights, that is, with $\alpha_j = \beta_j$ for each j $(j = 1, \ldots, n)$, is solvable in $O(n^2 \Sigma_{j=1}^n p_j)$ time and $O(n \Sigma_{j=1}^n p_j)$ space. We develop two pseudopolynomial dynamic programming algorithms, one for the small variant and one for the large variant, that are similar in structure, both taking advantage of the following properties that apply to either variant:

- any optimal schedule is compact;
- any optimal schedule has at least one job started before and completed at or after its due date; we call the *first* job of this kind the pivotal job, thereby stretching our previous definition;
- all jobs after the pivot are just-in-time or tardy, and, by definition of the pivotal job, all jobs before the pivot are early;
- any optimal schedule is V-shaped in the sense that all the jobs before the pivot appear in order of nondecreasing values α_j/p_j; all jobs after the pivot appear in order of nonincreasing values α_j/p_j.

These properties are easily checked and therefore go without proof. Both algorithms build on the dynamic programming algorithm for the large common due date problem with distinct symmetrical weights according to Hall and Posner[4] and on the dynamic programming algorithm for the small common due date problem with distinct symmetrical weights according to Hoogeveen and van de Velde[6]; both are therefore presented without much detail.

4.1. The Large Variant

This variant has the additional property that there is an optimal schedule in which the pivotal job is completed *exactly* on its due date. Since we do not know the identity of the pivotal job, we break the problem down into n subproblems, one for each tentative pivot. We reindex the jobs in order of nonincreasing α_j/p_j. For any pivotal job J_q, let $\mathcal{L}_j^q = \Sigma_{h=1,h\neq q}^j p_h$, and let $F_j^q(t)$ denote the minimal cost for scheduling $J_1, \ldots, J_j$ subject to the condition that the machine processes these jobs in the interval $[d_q - p_q - t, d_q + \mathcal{L}_j^q - t]$; the remaining unscheduled jobs must therefore be processed outside this interval.

Note that the cost of scheduling J_{j+1} $(J_{j+1} \neq J_q)$ depends on t only, not upon the identity of the early and tardy jobs. This gives rise to the following dynamic programming algorithm. The initialization is

$$F_j^q(t) = \begin{cases} 0, & \text{if } t = 0,\ j = 0, \\ \infty, & \text{otherwise,} \end{cases}$$

and the recursion for $j = 1, \ldots, n$, $t = 0, \ldots, \mathcal{X}_j^q$ is

$$F_j^q(t) = \begin{cases} F_{j-1}^q(t), & \text{if } j = q \\ F_{j-1}^q(t) + \alpha_j(d_q + \mathcal{X}_j^q - t - d_j), & \\ \qquad \text{if } j \neq q \text{ and } t < p_j, & \\ F_{j-1}^q(t) + \alpha_j(d_q + \mathcal{X}_j^q - t - d_j), & \\ \qquad \text{if } j \neq q,\ t = p_j, \text{ and } d_q - p_q = d_j, & \\ \min\{F_{j-1}^q(t - p_j) + \alpha_j(t + d_j - p_j - d_q + p_q), & \\ \quad F_{j-1}^q(t) + \alpha_j(d_q + \mathcal{X}_j^q - t - d_j)\}, & \\ \qquad \text{if } j \neq q \text{ and } t > p_j \text{ or } d_q - p_q < d_j. & \end{cases}$$

The second equation reflects the case that J_j can be scheduled after J_q only. The third equation concerns a technicality: since J_q is the designated pivot, we can have no just-in-time job before J_q by definition. The last equation reflects the case that we can schedule J_j at either side of the pivot. For any pivotal job J_q the optimal solution value is equal to $\min_{0 \leq t \leq \mathcal{X}_n^q} F_n^q(t)$. The optimal solution value to the large almost common due date problem with distinct symmetrical weights is then equal to $\min_{1 \leq q \leq n, 0 \leq t \leq \mathcal{X}_n^q} F_n^q(t)$ and can be found in $O(n^2 \sum_{j=1}^n p_j)$ time and $O(n \sum_{j=1}^n p_j)$ space. The optimal solution is found by backtracing.

4.2. The Small Variant

The dynamic programming algorithm for the large variant is not applicable to the small variant, since the due dates may now be so small that a decision to schedule a certain job early may be a decision to let this job start before time zero.

At least one of the following two statements is true: there is an optimal schedule with some job completed exactly on its due date, or there is an optimal schedule with all jobs scheduled in the interval $[0, \sum_{k=1}^n p_k]$.

If the first statement is true, then the problem is solved by exactly the same dynamic programming algorithm as the large variant, albeit we have to make sure that no job is started before time zero. This is ascertained by computing the values $F_j^q(t)$ for values $0 \leq t \leq \min\{\mathcal{X}_j^q, d_q - p_q\}$ only.

If the second statement is true, then we only know that the pivotal job is executed *around* its due date, but we do not know its completion time. Consequently, we cannot compute the cost if we schedule the jobs in nonincreasing order of values α_j/p_j around the pivot. We can, however, compute the scheduling cost relative to the endpoints of the interval if we appoint the pivot, leaving it unscheduled for the time being, and first schedule the other jobs in order of nondecreasing values α_j/p_j; after all, the nonpivotal job with *smallest* value α_j/p_j must be completed first or last.

For any given pivot J_q, let $P = \sum_{j=1}^n p_j$, $\mathcal{R}_j^q = \sum_{h=j,h\neq q}^n p_h$, and for any t ($t = \max\{0, d_q + \mathcal{R}_j^q - P\}, \ldots, d_q - 1$), let $G_j^q(t)$ be the minimum cost for scheduling $J_j, \ldots, J_n$ subject to the condition that the machine processes the nonpivotal jobs among $J_j, \ldots, J_n$ in the intervals $[0, t]$ and $[t + P - \mathcal{R}_j^q, P]$. The condition on t is imposed to reserve room for scheduling J_q around its due date. The initialization is

$$G_j^q(t) = \begin{cases} 0, & \text{if } t = 0,\ j = 0, \\ \infty, & \text{otherwise,} \end{cases}$$

and the recursion for $j = n, \ldots, 1$, $t = \max\{d_q + \mathcal{R}_j^q - P, 0\}, \ldots, d_q - 1$ is

$$G_j^q(t) = \begin{cases} G_{j-1}^q(t), & \text{if } j = q, \\ G_{j-1}^q(t) + \alpha_j(t + P - \mathcal{R}_j^q - d_j), & \\ \qquad \text{if } j \neq q \text{ and } t \geq d_j, & \\ \min\{G_{j-1}^q(t - p_j) + \alpha_j(d_j - t), & \\ \quad G_{j-1}^q(t) + \alpha_j(t + P - \mathcal{R}_j^q - d_j)\}, & \\ \qquad \text{if } j \neq q \text{ and } t < d_j. & \end{cases}$$

The second equation is due to our definition of pivot: if we would allow processing up to time $t \geq d_j$, then by definition J_q cannot be the pivot anymore. If $t < d_j$, then any job other than J_q can be scheduled on either side of J_q; this is expressed in the third equation. The recursion leaves the interval $[t, t + p_q]$ for each t with $\max\{d_q - p_q, 0\} \leq t \leq d_q - 1$ idle, and it is here that we insert the pivotal job J_q. We compute the resulting cost, which be denote by $G^q(t)$, as

$$G^q(t) = G_n^q(t) + \alpha_q(t + p_q - d_q),$$
$$\text{for } \max\{d_q - p_q, 0\} \leq t \leq d_q - 1.$$

For any given pivotal job J_q, the optimal solution value is then equal to $\min_{d_q - p_q \leq t \leq d_q - 1} G^q(t)$. Hence, the optimal solution value for scheduling the jobs in the interval $[0, P]$ is then equal to $\min_{1 \leq q \leq n, d_q - p_q \leq t \leq d_q - 1} G^q(t)$, which is found in $O(n^2 \sum_{j=1}^n p_j)$ time and $O(n \sum_{j=1}^n p_j)$ space.

The optimal solution value to the small almost common due date problem with distinct symmetrical weights is then equal to

$$\min\left\{ \min_{1 \leq q \leq n, 0 \leq t \leq \min\{\mathcal{X}_j^q, d_q - p_q\}} F_n^q(t), \min_{1 \leq q \leq n, \max\{d_q - p_q, 0\} \leq t \leq d_q - 1} G^q(t) \right\},$$

and the optimal solution is found by backtracing.

5. Related Large Due Date Problems

The principle of the dynamic programming algorithm described in Section 3 is applicable to any earliness–tardiness problem for which the objective function can be formulated in terms of positional costs, for which the optimal schedule is compact, and for which there is an optimal schedule that can be characterized by the pivotal job and the set of early jobs.

For large common due date problems, the dynamic programming algorithm takes the following generic form. Let $F_j(i)$ be the minimum cost for scheduling jobs $J_1, \ldots, J_j$ subject to the condition that i jobs among them are scheduled early or just-in-time. This corresponds to the large almost equal due date problem with $i - 1$ jobs scheduled *before* the pivot. The initialization is then

$$F_j(i) = \begin{cases} 0, & \text{if } j = 0,\ i = 0, \\ \infty, & \text{otherwise,} \end{cases}$$

and the recursion for $j = 1, \ldots, n$, $i = 0, \ldots, j$ is

$$F_j(i) = \min\{F_{j-1}(i-1) + \alpha e_{ji},\ F_{j-1}(i) + \beta t_{j,j-i}\},$$

where e_{ji} is the cost of assigning J_j to the ith early position, and $t_{j,j-i}$ is the cost of assigning J_j to the $(j-i)$th tardy position; by default $e_{j0} = \infty$ and $t_{j0} = \infty$. The optimal solution value is then $\min_{0 \le i \le n} F_n(i)$, and the optimal solution is found by backtracing. The dynamic programming algorithm requires $O(n^2)$ time and space.

Lee, Liman Danasaputra, and Lin[10] consider the large common due date problem of minimizing

$$f(\sigma) = \sum_{j=1}^{n} [\alpha E_j + \beta T_j + \gamma U_j],$$

where U_j $(j = 1, \ldots, n)$ is an indicator variable: $U_j = 1$ if J_j is tardy, and $U_j = 0$ otherwise. They solve this problem in $O(n^2 \log n)$ time. Even if the common weight γ is replaced with distinct weights γ_j, however, the objective function can be reformulated in terms of positional costs by letting $e_{ji} = \alpha(i-1)p_j$ and $t_{j,j-i} = \beta(j-i)p_j + \gamma_j$. Hence, the problem can be solved in $O(n^2)$ time. In case of large almost equal due dates, we have that $e^q_{ji} = \alpha((i-1)p_j + d_j - d_q)$ and $t^q_{j,j-i} = \beta((j-i)p_j + d_q - d_j) + \gamma_j$. We solve this problem in $O(n^3)$ time through the dynamic programming algorithm of Section 2.

Another problem solvable by the dynamic programming algorithm is the large common due date problem of minimizing

$$f(\sigma) = \sum_{j=1}^{n} [\alpha E_j + \beta T_j] + \gamma T_{\max},$$

where $T_{\max}$ is defined as $T_{\max} = \max_{1 \le j \le n} T_j$. Because of the due date structure, $T_{\max}$ is equal to the tardiness of the last job in the schedule. The objective function can be reformulated in terms of positional costs by letting $e_{ji} = \alpha(i-1)p_j$ and $t_{j,j-i} = \beta(j-i)p_j + \gamma p_j = (\beta(j-i) + \gamma)p_j$; it is solved in $O(n \log n)$ time by applying Emmons' matching algorithm. In case of large almost equal due dates, $T_{\max}$ is again attained by the last job in the schedule; hence, $T_{\max} = C_{[n]} - d_{[n]} = d_q + \sum_{h \in \mathcal{T}} p_h - d_{[n]}$, where $\mathcal{T}$ denotes the set of jobs that are scheduled after the pivot J_q. We have that $e^q_{ji} = \alpha((i-1)p_j + d_j - d_q)$, $t^q_{j,j-i} = \beta((j-i)p_j + d_q - d_j) + \gamma p_j$ for $i = 0, \ldots, j-2$, and $t^q_{j,1} = \beta(p_j + d_q - d_j) + \gamma(p_j + d_q - d_j)$. We solve this problem in $O(n^3)$ time by the dynamic programming algorithm of Section 2.

Finally, we consider the large common due date problem of minimizing

$$f(\sigma) = \sum_{j=1}^{n} [\alpha E_j + \beta T_j + \gamma C_j].$$

Table I. Overview of Time Complexities

Variant	Common due date	Almost common due date
large, $\alpha_j = \alpha$, $\beta_j = \beta$	$O(n \log n)$	$O(n^2)$
large, $\alpha_j = \beta_j$	$O(n \sum_{j=1}^n p_j)$	$O(n^2 \sum_{j=1}^n p_j)$
small, $\alpha_j = \beta_j$	$O(n^2 \sum_{j=1}^n p_j)$	$O(n^2 \sum_{j=1}^n p_j)$
general	open	open

If $\gamma \ge \alpha$, then all jobs must be processed in the interval $[0, \sum_{j=1}^n p_j]$, and the jobs must be executed in order of nondecreasing processing times. If $\gamma < \alpha$, then the dynamic programming algorithm is applicable if we manage to express $\sum_{j=1}^n C_j$ in terms of positional costs. Obviously, there is an optimal compact schedule σ^* in which some job is completed at its due date; the start time of $J_{[1]}$, that is, the first job in σ^*, is then equal to $d - \sum_{J_j \in \mathcal{E}} p_j$, where $\mathcal{E}$ denotes the set of early jobs in σ^*. Consequently, the completion time of $J_{[k]}$, for any $k = 1, \ldots, n$, is equal to $(d - \sum_{J_j \in \mathcal{E}} p_j + \sum_{h=1}^k p_{[h]})$. Hence, if we initialize the recursion as

$$F_j(i) = \begin{cases} \gamma n d, & \text{if } j = 0,\ i = 0, \\ \infty, & \text{otherwise,} \end{cases}$$

then we can apply the dynamic programming algorithm by putting $e_{ji} = \alpha(i-1)p_j - \gamma(i-1)p_j = (i-1)(\alpha - \gamma)p_j$ and $t_{j,j-i} = \beta(j-i)p_j + \gamma(j-i)p_j = (j-i)(\beta + \gamma)p_j$ and solve the problem in $O(n^2)$ time.

In case of large almost equal due dates, we have that $e^q_{ji} = (i-1)(\alpha - \gamma)p_j + \alpha(d_j - d_q)$ and $t^q_{j,j-i} = (j-i)(\beta + \gamma)p_j + \beta(d_q - d_j)$, and we solve the problem in $O(n^3)$ by the dynamic programming algorithm of Section 2.

6. Conclusion

We have identified a more general type of due date structure that gives rise to earliness–tardiness problems that, just like the common due date problems, are structurally less complicated than the problems with general due dates. Table I provides a succinct overview of the complexity of the different variants of the almost common due date problem and their common due date counterparts.

Future research on almost common due date problems may take two directions. First, it is yet unknown whether a speed-up for the related almost common due date problems discussed in Section 5 is possible. Second, Lagrangian relaxation gives rise to successful optimization and approximation algorithms for the common due date problem.[5] The question remains whether it is equally successful for the almost common due date problem.

Acknowledgment

The authors thank two anonymous referees for their helpful comments on an earlier draft of the paper. The research of the first author was supported by a grant from Netherlands Organization for Scientific Research.

Appendix A: Proof of Theorem 3

Theorem 3. *There exists an optimal schedule for the almost common due date problem with some pivotal job J_q in which each job J_j with $j > q$ is scheduled before the pivot if and only if*

$$d_j - p_j \leq d_q - p_q,$$

or

$$d_j < d_q \quad \text{and} \quad \alpha(d_j - p_j - d_q + p_q) \leq \beta(d_q - d_j).$$

Proof. Let σ be any optimal schedule and let J_q be its pivot. First, consider the case that there exists some other job J_c that is just-in-time; without loss of generality, we suppose that $c < q$. We prove that either all jobs J_j with $j > c$ are scheduled as dictated by Theorem 3, or σ can be transformed to a schedule with equal cost in which only J_q is just-in-time. Because of the due date structure and because J_c and J_q are both just-in-time, we must have that $d_c = d_q - p_q$ or $d_c - p_c = d_q$. Suppose that we have $d_c = d_q - p_q$; J_c is executed in σ immediately before J_q. Let J_j be any job with $j > c$, implying $p_j \leq p_c$. We must have $d_j \geq d_c$; otherwise, $d_j < d_c = d_q - p_q$, which contradicts our assumption about the almost equal due dates. Combining $d_j \geq d_c$ and $p_j \leq p_c$ yields $d_j - p_j \geq d_c - p_c$. Equality holds if both $p_j = p_c$ and $d_j = d_c$. Because we can interchange two identical jobs without changing the schedule, we may assume that J_c has the highest index among these identical jobs. Therefore, we may assume that $d_j - p_j > d_c - p_c$; hence, Theorem 3 is satisfied if J_j is executed after J_c in σ. Since σ has to be V-shaped to be optimal, J_j will be executed after J_c, unless $p_c = p_j$. Interchanging J_c with the highest indexed job J_j with $j > c$ and $p_j = p_c$ that is scheduled before J_c does not change the cost; since the due date of the jobs interchanged with J_c is larger than d_c, this job will not be just-in-time, which implies that J_q is the only just-in-time job in this schedule. The case $d_c - p_c = d_q$ can be dealt with in the same way.

Hence, we may now assume that J_q is the only just-in-time job in σ. We prove that, unless all jobs J_j with $j > q$ are scheduled as dictated by Theorem 3, σ can be transformed into either a schedule $\bar{\sigma}$ with $f(\bar{\sigma}) < f(\sigma)$, contradicting the optimality of σ, or into a schedule $\hat{\sigma}$ with $f(\hat{\sigma}) = f(\sigma)$ in which a higher indexed job serves as pivot. This approach eventually yields a schedule that satisfies Theorem 3, since any optimal schedule with pivot J_n obviously satisfies Theorem 3.

Suppose that J_j is the smallest indexed job that is not scheduled as dictated by Theorem 3. There are four cases to consider:

1. $d_j - p_j \leq d_q - p_q$ and J_j tardy;
2. $d_j - p_j > d_q - p_q$ and
   a. $d_j < d_q$, $\alpha(d_j - p_j - d_q + p_q) \leq \beta(d_q - d_j)$, and J_j tardy;
   b. $d_j < d_q$, $\alpha(d_j - p_j - d_q + p_q) > \beta(d_q - d_j)$, and J_j early;
   c. $d_j \geq d_q$ and J_j early.

For each case, let $\bar{\sigma}$ be the compact schedule that results from σ by interchanging J_q and J_j and retaining the start time of the first job in σ. The interchange affects then only J_q, J_j, and the jobs in between, that is, the job set

$$\mathfrak{X} = \{J_h | \min\{C_j(\sigma), C_q(\sigma)\} < C_h(\sigma) < \max\{C_j(\sigma), C_q(\sigma)\}\};$$

hence, in order to evaluate $f(\sigma) - f(\bar{\sigma})$, it suffices to evaluate the cost of J_q, J_j, and the jobs in $\mathfrak{X}$.

Consider case 1: $d_j - p_j \leq d_q - p_q$ and J_j tardy. Then $C_j(\bar{\sigma}) = d_q - p_q + p_j \geq d_j$. Consequently, J_j is tardy or just-in-time; because of the due date structure, all jobs after J_j are tardy or just-in-time. Since the interchange has not created any early jobs, we know that the total tardiness of J_q, J_j, and the jobs in $\mathfrak{X}$ in σ and $\bar{\sigma}$ is equal to their total completion time minus the sum of their due dates. Because $C_h(\bar{\sigma}) = C_h(\sigma) + p_j - p_q \leq C_h(\sigma)$ for all $J_h \in \mathfrak{X}$, we have that $f(\bar{\sigma}) \leq f(\sigma)$. More precisely, we have that $f(\bar{\sigma}) < f(\sigma)$, unless $p_q = p_j$. Recall that J_q has index lower than J_j if and only if $p_q > p_j$ or $p_q = p_j$ and $d_q \leq d_j$; hence, in case $p_q = p_j$, we have $d_q \leq d_j$. Combining this inequality with $d_j - p_j \leq d_q - p_q$ and $p_q = p_j$ yields $d_q = d_j$. This implies that J_j is just-in-time and serves as pivot in $\bar{\sigma}$.

Consider case 2.a: $d_j - p_j > d_q - p_q$, $d_j < d_q$, $\alpha(d_j - p_j - d_q + p_q) \leq \beta(d_q - d_j)$, and J_j tardy. Note that the combination of the first two inequalities implies that p_j is strictly smaller than p_q.

If $\mathfrak{X} = \emptyset$, then $f(\sigma) - f(\bar{\sigma}) = \beta(d_q - d_j) - \alpha(d_j - p_j - d_q + p_q) \geq 0$. Since the completion times of all other jobs remain equal, we have that no job in $\bar{\sigma}$ is just-in-time. Let $\hat{\sigma}$ be the schedule obtained by deferring all jobs in $\bar{\sigma}$ by $E_j(\bar{\sigma})$ units of time: J_j is then completed exactly at its due date and serves as pivot in $\hat{\sigma}$. We must have $f(\hat{\sigma}) \leq f(\bar{\sigma})$. After all, if $f(\hat{\sigma}) > f(\bar{\sigma})$, which means that the shifting has increased the cost, then *putting $\bar{\sigma}$ earlier* would render a schedule with cost smaller than $f(\bar{\sigma}) = f(\sigma)$, contradicting the optimality of σ.

Suppose that $\mathfrak{X} \neq \emptyset$ and that all jobs in $\mathfrak{X}$ remain tardy in $\bar{\sigma}$. Since $p_j < p_q$, we then have that $f(\sigma) - f(\bar{\sigma}) > 0$, which contradicts the optimality of σ. The only case remaining has z jobs ($z > 0$) in $\mathfrak{X}$ that are early in $\bar{\sigma}$. Let n_E and n_T denote the number of early and tardy jobs in σ. Since σ is optimal, putting the schedule earlier by one unit of time does not decrease $f(\sigma)$, implying that $\alpha(n_E + 1) \geq \beta n_T$. There are $(n_E + z + 1)$ early and at most $(n_T - z)$ tardy jobs in $\bar{\sigma}$. Deferring $\bar{\sigma}$ by one unit of time decreases the cost of the whole schedule by at least $[\alpha(n_E + z + 1) - \beta(n_T - z)] \geq z(\alpha + \beta)$. Let $\{E_{[1]}, \ldots, E_{[z]}\}$ denote the earlinesses of the z early jobs in $\bar{\sigma}$, where $E_{[h]} \leq E_{[h+1]}$ for $h = 1, \ldots, z - 1$. Deferring $\bar{\sigma}$ by $E_{[1]}$ units of time decreases its cost by $z(\alpha + \beta)E_{[1]}$; in this schedule, $z - 1$ jobs in $\mathfrak{X}$ are early. Deferring this schedule by another $E_{[2]} - E_{[1]}$ units of time decreases its cost by $(z - 1)(\alpha + \beta)(E_{[2]} - E_{[1]})$ and leaves $(z - 2)$ jobs in $\mathfrak{X}$ early. Let $\hat{\sigma}$ denote the schedule obtained by shifting $\bar{\sigma}$ by $E_{[z]}$ units of time. From the above analysis, it follows that $f(\hat{\sigma}) \leq f(\bar{\sigma}) - \Delta$, where

$$\Delta = (\alpha + \beta)\{zE_{[1]} + (z - 1)(E_{[2]} - E_{[1]}) + \cdots + (E_{[z]} - E_{[z-1]})\}$$

$$= (\alpha + \beta) \sum_{h \in \mathfrak{X}} E_h > \alpha \sum_{h \in \mathfrak{X}} E_h.$$

Because $\Sigma_{h \in \mathfrak{X}} T_h(\bar{\sigma}) \leqslant \Sigma_{h \in \mathfrak{X}} T_h(\sigma)$, and because the cost of J_h and J_j in $\bar{\sigma}$ is no more than their cost in σ, we have that $f(\hat{\sigma}) < f(\sigma)$, contradicting the optimality of σ.

Consider case 2.b: $d_j - p_j > d_q - p_q$, $d_j < d_q$, $\alpha(d_j - p_j - d_q + p_q) > \beta(d_q - d_j)$, and J_j early. The analysis of this case proceeds along the same lines as the analysis of case 2.a.

Consider case 2.c: $d_j \geqslant d_q$ and J_j early. Because $C_j(\bar{\sigma}) = C_q(\sigma) = d_q$, J_j will be early or just-in-time in $\bar{\sigma}$; because of the due date structure, J_q and all jobs in $\mathfrak{X}$ are early or just-in-time in $\bar{\sigma}$. If there is a job other than J_j that is just-in-time in $\bar{\sigma}$, then we must have $p_j < p_q$, since this job was not just-in-time in σ. In case of $p_j < p_q$, we have that $f(\bar{\sigma}) < f(\sigma)$, contradicting the optimality of σ. If $p_j = p_q$, then we have either that J_j is just-in-time and serves as a pivot in $\bar{\sigma}$, or that we can shift $\bar{\sigma}$ without increasing the cost such that we obtain a schedule $\bar{\sigma}$ in which J_j is just-in-time and serves as a pivot. ■

References

1. K. BAKER and G. SCUDDER, 1990. Sequencing with Earliness and Tardiness Penalties: A Review, *Operations Research 38,* 22–57.
2. H. EMMONS, 1987. Scheduling to a Common Due Date on Parallel Uniform Processors, *Naval Research Logistics 34,* 803–810.
3. N.G. HALL, W. KUBIAK, and S.P. SETHI, 1991. Earliness–Tardiness Scheduling Problems, II: Deviation of Completion Times about a Common Due Date, *Operations Research 39,* 847–856.
4. N.G. HALL and M.E. POSNER, 1991. Earliness–Tardiness Scheduling Problems, I: Weighted Deviation of Completion Times about a Common Due Date, *Operations Research 39,* 836–846.
5. J.A. HOOGEVEEN, H. OOSTERHOUT, and S.L. VAN DE VELDE, 1994. New upper and lower bounds for scheduling around a small common due date, *Operations Research 42,* 102–110.
6. J.A. HOOGEVEEN and S.L. VAN DE VELDE, 1991. Scheduling around a small common due date, *European Journal of Operational Research 55,* 237–242.
7. J.A. HOOGEVEEN and S.L. VAN DE VELDE, 1996. A branch-and-bound algorithm for single-machine earliness–tardiness scheduling with idle time, *INFORMS Journal on Computing.* (to appear).
8. J.J. KANET, 1981. Minimizing the average deviation of job completion times about a common due date, *Naval Research Logistics Quarterly 28,* 643–651.
9. E.L. LAWLER, 1977. A 'pseudopolynomial' algorithm for sequencing jobs to minimize total tardiness, *Annals of Discrete Mathematics 1,* 331–342.
10. C.-Y. LEE, S. LIMAN DANASAPUTRA, and C.-S. LIN, 1991. Minimizing weighted number of tardy jobs and weighted earliness–tardiness penalties about a common due date, *Computers and Operations Research 18,* 379–389.
11. J.K. LENSTRA, A.H.G. RINNOOY KAN, and P. BRUCKER, 1977. Complexity of machine scheduling problems, *Annals of Discrete Mathematics 1,* 343–362.