Optimal Timing Schedules in Earliness-Tardiness Single Machine Sequencing

Wlodzimierz Szwarc and Samar K. Mukhopadhyay
School of Business Administration, University of Wisconsin-Milwaukee, Milwaukee, Wisconsin

The article deals with a single machine earliness-tardiness scheduling model where idle times are permitted in job processing. Based on a cluster concept we develop properties of the model that lead to a very fast algorithm to find an optimal timing schedule for a given sequence of jobs. The performance of this algorithm is tested on 480 randomly generated problems involving 100, 200, 400 and 500 jobs. It takes less than two seconds to solve a 500 job problem on a PC. © 1995 John Wiley & Sons, Inc.

1. INTRODUCTION

Davis and Kanet [1] recently considered the general earliness-tardiness single machine problem (later called ET model) with arbitrary earliness and tardiness penalties where idle times are permitted in job processing. They developed a procedure, called Timetabler, to determine for a given sequence of jobs the optimal timing schedule which specifies their optimal start times. They applied the Timetabler in a branch and bound solution procedure. For an extensive discussion of the ET model see Davis and Kanet [1]. One should mention two papers not listed in Davis and Kanet [1] that also developed optimal timing algorithms: by Garey, Tarjan, and Wilfong [2] for the case when the earliness and tardiness penalties are equal, and Yano and Kim [3] for the case when the tardiness penalties are not smaller than the earliness penalties. Garey, Tarjan, and Wilfong [2] managed to reduce the complexity of their algorithm from $O(n^2)$ to $O(n \log n)$, where n is the number of jobs.

We identify certain structural properties of optimal timing schedules for a given sequence of jobs and show how to split this sequence into a number of special subsequences, called *clusters,* where processing goes on uninterrupted. Thus idle time may occur *only* between clusters. The cluster decomposition does not depend on the earliness or tardiness penalties. Neither does it depend on the start time of the cluster. We prove that in each cluster the early jobs *always* precede the non-early jobs. Moreover, the earliness of the jobs does not increase while their tardiness does not decrease down the sequence. Taking advantage of these and other properties of clusters we design a simple and very efficient optimal timing algorithm of complexity $O(mn)$ where m is the number of clusters. The basic difference between our procedure and the known algorithms is that we schedule blocks of clusters whereas the other authors schedule single jobs, or at best blocks of jobs that do not enjoy the elegant properties of clusters. The main feature of our algorithm is that the emerging idle times (between clusters) *do not change during the procedure,* thereby increasing its efficiency. If the entire sequence forms a single cluster or n clusters, then the optimal timing schedule is instantly found. Our algorithm tested on 480 random problems on a PC proves to be very fast. It takes less than 2 seconds on average to solve a 500 job

Naval Research Logistics, Vol. 42, pp. 1109–1114 (1995)

CCC 0894-069X/95/071109-06

problem. The developed results should be useful in improving the efficiency of any branch and bound solution procedure of the ET model.

2. NOTATION

We will use the following notations:
p_k—the processing time of job k, $k \in I$,
d_k—due date of job k,
α_k—earliness penalty of job k,
β_k—tardiness penalty of job k.
Throughout the paper we will deal only with a single sequence $S = 1, 2, \ldots, n$.
C_k—completion time of job k in S.
$C = (C_1, C_2, \ldots, C_n)$, vector of completion times, called schedule.
$E_k = d_k - C_k$, if $d_k > C_k$, earliness of job k,
$T_k = C_k - d_k$, if $d_k \leq C_k$, tardiness of job k.
The problem can be stated as follows: Find a schedule C that minimizes

$$f(C) = \sum_{k=1}^{n} (\alpha_k E_k + \beta_k T_k). \tag{1}$$

3. BASIC PROPERTIES OF THE ET MODEL

First we explore when there is idle time between two adjacent jobs in an optimal schedule. We prove the following property.

PROPERTY 1: If there is any idle time between (consecutive) jobs i and $i + 1$ in an optimal schedule, then $d_{i+1} - d_i > p_{i+1}$.

PROOF: In the optimal schedule job i cannot be early ($C_i \geq d_i$) and job $i + 1$ cannot be late ($C_{i+1} \leq d_{i+1}$). Then $C_{i+1} > C_i$ imply that $d_{i+1} - d_i \geq C_{i+1} - C_i$. Since there is idle time, we have $C_{i+1} - C_i > p_{i+1}$, and therefore $d_{i+1} - d_i > p_{i+1}$. □

Consider the following condition:

$$d_{i+1} - d_i \leq p_{i+1}. \tag{2}$$

Let $\sigma = u, u + 1, \ldots, v$, $u < v$, be a subsequence of $S = 1, 2, \ldots, n$.

Sequence σ is called a *cluster* if (2) holds for each $u \leq i \leq v - 1$ and does not hold for $i = u - 1$ and $i = v$. Obviously, jobs i and $i + 1$ belong to different clusters if (2) is not satisfied. By verifying (2) for each i we decompose S into a sequence of single or multijob clusters $\sigma_1, \sigma_2, \ldots, \sigma_m$. Consider the following example of Davis and Kanet [1].

EXAMPLE 1:

$p_1, \ldots, p_7 = 3, 2, 7, 3, 6, 2, 8$; $d_1, \ldots, d_7 = 12, 4, 26, 18, 16, 25, 30$;

$\alpha_1, \ldots, \alpha_7 = 10, 20, 18, 9, 10, 16, 11$; $\beta_1, \ldots, \beta_7 = 12, 25, 38, 12, 12, 18, 15$.

Using (2) we decompose $P = 1, 2, \ldots, 7$ into $\sigma_1, \sigma_2, \sigma_3$ where $\sigma_1 = 1, 2$, $\sigma_2 = 3, 4, 5$, and $\sigma_3 = 6, 7$.

Property 1 implies the following.

PROPERTY 2: In an optimal timing schedule the jobs of each cluster are processed without interruption.

Let C be an optimal timing schedule where jobs i and $i + 1$ belong to cluster σ and t is the start time of job i. Adding $-t - p_i$ to both sides of inequality (2) written as $d_{i+1} - p_{i+1} \le d_i$ we get $d_{i+1} - t - p_i - p_{i+1} \le d_i - t - p_i$ which can be written as

$$d_i - C_i \ge d_{i+1} - C_{i+1}. \tag{3}$$

The implications of (3) can be summarized by the following property.

PROPERTY 3: In each cluster of an optimal schedule the early jobs precede the non-early jobs. Moreover $E_i \ge E_{i+1}$ if jobs i and $i + 1$ of the same cluster are early, while $T_i \le T_{i+1}$ if those jobs are not early.

4. THE OPTIMAL TIMING ALGORITHM

The solution procedure starts out with an initial timing schedule $C = (C_1, C_2, \ldots, C_n)$ for sequence $S = 1, 2, \ldots, n$ where $C_i = \sum_{k=1}^{i} p_k$ for each $i = 1, 2, \ldots, n$.

Next decompose S via (2) into clusters $\sigma_1, \sigma_2, \ldots, \sigma_m$. Since the C_i are the *earliest possible* completion times of the jobs we have to determine how much to *increase* (shift) those C_i to get the optimal completion times for each job. In fact, it is sufficient to calculate the shift in each cluster since all its jobs are shifted by the *same* amount.

From now on we only keep track of early jobs in each cluster $\sigma_r = u, u + 1, \ldots, v$ of the initial schedule C. To find those jobs we take advantage of (3) and check condition $d_k - C_k > 0$ starting from $k = u, u + 1, \cdots$ until it no longer holds. Next calculate $\Delta_k = \sum_{s=u}^{k} \alpha_s - \sum_{s=k+1}^{v} \beta_s$, which can be expressed as

$$\Delta_k = \Delta_{k-1} + \alpha_k + \beta_k, \qquad \Delta_0 = -\sum_{k=u}^{v} \beta_k. \tag{4}$$

Define a block as a sequence of clusters processed without interruption. Consider block $\sigma_s, \sigma_{s+1}, \ldots, \sigma_m$.

Let j_r be the *last early* job of σ_r. Suppose that $d_{j_r} - C_{j_r} \ge 1$ for every last job of cluster σ_r, $s \le r \le m$. It is easy to see that the shift of the entire block by one unit decreases the cost of the schedule by $\sum_{r=s}^{m} \Delta_r$.

Define $E(r) = d_{j_r} - C_{j_r}$, $\Delta(r) = \Delta_{j_r}$. Then

$$E(r) = \min_{u \le k \le j_r} (d_k - C_k), \qquad \Delta(r) = \max_{u \le k \le j_r} \Delta_k. \tag{5}$$

If none of the jobs of σ_r is early, then

$$E(r) = \infty, \qquad \Delta(r) = \Delta_0. \tag{6}$$

The idea of the optimal timing algorithm is to identify the first block of clusters that cannot be shifted. This block stays "behind." The procedure is repeated for the reduced set of clusters to identify the next block that cannot be shifted. Otherwise all *remaining* clusters are shifted by the smallest $E(r)$ such that a last early job of a cluster becomes an on-time job. This procedure is repeated again after all C_k are updated, and all non-early jobs are removed from the list for each cluster. The algorithm stops once a block involving the last cluster σ_m cannot be shifted. The additional advantage of this algorithm is that it does not explicitly deal with idle times. In our procedure they emerge automatically once a block of clusters stays behind while all remaining clusters are pushed forward. Since the idle times *do not change during the procedure,* the size of the problem keeps reducing, making the algorithm very efficient.

The optimal timing algorithm can be stated as follows.

Step 0: Identify the clusters and calculate the respective $\Delta(r)$.

Step 1: Find the smallest s such that $\sum_{r=1}^{s} \Delta(r) \le 0$. Set the original C_k for each job of the first s clusters. If $s = m$, Stop. Else go to Step 2. If no such s exists, go to Step 3.

Step 2: Remove the first s clusters from the list, reindex the remaining jobs, and go to Step 1 to consider the reduced set of clusters.

Step 3: Find $\min_{1 \le r \le m} E(r)$. Increase all C_k by $\min E(r)$. Eliminate all last early jobs of clusters that are no longer early. Update (5) and (6). Go to Step 1.

Our algorithm produces a schedule that cannot be improved by shifting a single job (or a single cluster). According to Davis and Kanet [1] such a schedule is optimal.

We illustrate our algorithm by solving Example 1 where $\sigma_1 = (1, 2)$, $\sigma_2 = (3, 4, 5)$ and $\sigma_3 = (6, 7)$. The initial schedule $C = (C_1, \ldots, C_7) = (3, 5, 12, 15, 21, 23, 31)$. Then (1), (3, 4) and (6) are the sets of early jobs of clusters σ_1, σ_2 and σ_3. Calculate $d_k - C_k$ for all these jobs. Next use Eq. (5) to find $E(1) = 9$, $E(2) = 3$, $E(3) = 2$, $\Delta(1) = -15$, $\Delta(2) = 15$, $\Delta(3) = 1$.

Step 1: $s = 1$ since $\Delta(1) < 0$. Cluster σ_1 is not shifted. Set $C_1 = 3$, $C_2 = 5$.

Step 2: Eliminate σ_1.

Step 1: $\Delta(2) > 0$, $\Delta(2) + \Delta(3) > 0$.

Step 3: $\min[E(2), E(3)] = \min(3, 2) = 2$. Increase all C_i, $3 \le i \le 7$ by 2 units. Eliminate job 6 from the list of early jobs. Update $d_k - C_k$. Use Eq. (5) and (6) to find $E(2) = 1$, $E(3) = \infty$, $\Delta(2) = 25$, $\Delta(3) = -33$.

Step 1: $\Delta(2) > 0$, $\Delta(2) + \Delta(3) < 0$, $s = 3 = m$. Stop. Clusters σ_2 and σ_3 are not shifted.

The optimal schedule $S_{\text{opt}} = (3, 5, 14, 17, 23, 25, 33)$.

Based on our algorithm we offer rules to instantly solve cases when $m = 1$ or $m = n$. Their complexity is $0(n)$.

RULE 1: If S is a single cluster and j is its last early job, calculate Δ_k for each $k = 1, 2, \cdots$ and find a job with the *smallest* index i, $i \le j$, satisfying condition $\Delta_k > 0$. Then $S_{\text{opt}} = (C_1 + d_i - C_i, \ldots, C_n + d_i - C_i)$. If no such i exists, then $S_{\text{opt}} = (C_1, \ldots, C_n)$.

Table 1. Performance of the optimal timing algorithm average CPU time per example (sec).

	$n = 100$		$n = 200$		$n = 400$		$n = 500$	
a–b	D–K	ours	D–K	ours	D–K	ours	D–K	ours
0.1–0.9	2.80	0.96	9.37	0.98	34.63	1.68	53.83	1.92
0.2–0.8	2.95	0.78	10.24	0.99	36.70	1.55	57.49	1.79
0.3–0.6	3.22	0.79	10.93	1.14	39.59	1.80	61.40	1.76
0.1–1.3	2.51	1.49	8.26	1.34	32.10	1.64	45.23	1.76
0.1–1.7	2.29	1.48	11.85	1.67	29.46	1.52	40.73	1.67
0.1–2.1	2.22	1.35	10.56	1.60	27.48	1.59	39.34	1.82
Overall average	2.66	1.14	10.20	1.29	33.33	1.63	49.67	1.79

RULE 2: Suppose every job of S is a *separate* cluster (then $d_{i+1} - d_i > p_{i+1}\ \forall\ i \le n-1$) and k is its first early job in the schedule with no inserted idle time. Then $C_{\text{opt}} = (\bar{C}_1, \ldots, \bar{C}_n)$ where $\bar{C}_i = C_i$ for $i < k$ and $\bar{C}_i = d_i$ for $i \ge k$.

5. COMPUTATIONAL EXPERIENCE

We tested our method on 480 randomly generated problems on IBM PS/2-70 using PASCAL. The integer processing times were drawn from a uniform distribution in a range [1, 100], the integer earliness and tardiness penalties were drawn from a uniform distribution in a range [1, 10] while the due dates were generated from a uniform distribution of integer in a range [pa, pb] where $p = \sum_{k=1}^{n} p_k$. Then tardiness factor $T = 1 - [(a+b)/2]$ and the relative range is $R = b - a$. For six combinations of a, b twenty test problems were generated for $n = 100$, 200, 400 and 500. The results for those 480 problems are summarized in Table 1. We also tested Davis and Kanet algorithm (labeled D–K).

The results in Table 1 indicate that our algorithm is very fast. It takes less than two seconds to solve a 500 job problem. We also compare the average CPU performance of the two algorithms as n increases from 100 to 500. According to Table 1, the average CPU time of the D–K algorithm increases 18.65 times while this time for our algorithm increases by only 56%. It is interesting to note that the average number of clusters is relatively close to $n/2$. To explore why the CPU time of our algorithm is practically linear, we register the

Table 2. Separate frequencies of steps 2 and 3 per example in a 20 example run.

	$n = 100$		$n = 200$		$n = 400$		$n = 500$	
	Ave 2	Max 2	Ave 2	Max 2	Ave 2	Max 2	Ave 2	Max 2
a–b	Ave 3	Max 3	Ave 3	Max 3	Ave 3	Max 3	Ave 3	Max 3
0.1–0.9	2.5	9	19.25	32	64	84	87.2	108
	2.1	12	0	0	0	0	0	0
0.2–0.8	3.2	13	16.55	25	66.2	89	91.1	109
	1.6	11	0	0	0	0	0	0
0.3–0.6	6.35	17	17.6	26	77.05	97	103.25	116
	0.45	7	0	0	0	0	0	0
0.1–1.3	1.5	5	4.15	13	28.25	54	36.6	55
	13.85	24	4.1	18	0.5	1	0.5	1
0.1–1.7	2.45	6	4.3	11	13.65	22	19.75	34
	15.75	26	8.05	21	1.25	6	0.75	3
0.1–2.1	3.05	8	3.6	7	6.95	13	12.05	22
	16.4	28	7.4	17	1.55	5	1	2

number of times Steps 2 and 3 are executed in a 20 example run. At each iteration of our algorithm either a block of clusters is eliminated from being shifted (Step 2) or all considered clusters are shifted (Step 3). Step 2 is computationally negligible since it does not require any additional computation while after the shift in Step 3 C_k, $E(r)$ and $\Delta(r)$ are updated. Table 2 provides the average and maximum frequency per example of Steps 2 and 3 in a 20 example run. Step 3 is not executed in 262 examples including *all* 180 examples for $n \geq 200$ and $b < 1$. Thus processing in those schedules starts at zero and goes on uninterrupted. The number of times Step 3 is executed (i.e., the number of shifts) drastically decreases as n increases. The maximum number of shifts in an example is 28. For Step 2 the maximum number of shifts is 116 (in a 500 job example).

We also tested the performance of our algorithm on the same 480 examples for the special case where all earliness and tardiness penalties are 1. The average CPU times improved by 19.7%, 30.6%, 20.8% and 16.6% for $n = 100$, 200, 400 and 500, respectively.

REFERENCES

[1] Davis, J.S., and Kanet, J.J.. "Single-Machine Scheduling with Early and Tardy Completion Costs," *Naval Research Logistics,* **40,** 85–101 (1993).

[2] Garey, M.R., Tarjan, R.E., and Wilfong, G.T., One-Processor Scheduling with Symmetric Earliness and Tardiness Penalties," *Mathematics of Operations Research,* **13,** 330–348 (1988).

[3] Yano, C.A., and Kim, Y-D., Algorithms for a Class of Single-Machine Weighted Tardiness and Earliness Problems," *European Journal of Operational Research,* **52,** 167–178 (1991).