MINMAX EARLY-TARDY SCHEDULING WITH A COMMON DUE-DATE

Jayabrata Ghosh[1)]

[1)]Korea University Business School, Seoul, South Korea

Abstract

The global diffusion of the Just-in-Time philosophy of production in industry has prompted scheduling researchers to pay attention to the minimization of job earliness in addition to job tardiness. Much work has been done thus far. The research objective mostly is to minimize the sum of the earliness and tardiness penalties. However, work has also been done toward minimizing the maximum of these penalties (which is more relevant in certain situations). It has been shown that, even for a single machine and a common due-date for all jobs, the problem is very difficult to solve if the unit penalties are job-specific. In this paper, we revisit the problem and focus on the single machine, common due-date version. We present both an exact (yet practical) solution for a special case (which is also difficult) and an efficient heuristic solution for the general case. We make some observations on the extensibility of our work.

1. Introduction

Suppose that there is a single machine available at time 0, which can process jobs continuously and indefinitely, but only one at a time. There is also a set of n independent jobs, numbered 1 through n, that are ready at this time and must be processed on this machine without interruption. Let job j have a processing time p_j, a unit earliness penalty α_j and a unit tardiness penalty β_j. Finally, let d be the common due-date for all the n jobs, and $c_j(\sigma)$ be the completion time of job j in some schedule σ. Now, define $e_j(\sigma) = \max\{0, d - c_j(\sigma)\}$ and $t_j(\sigma) = \max\{0, c_j(\sigma) - d\}$ to be, respectively, the earliness and the tardiness of job j in schedule σ. The objective is to find a schedule which has the minimum value, among all schedules, for the maximum of the weighted earliness or the weighted tardiness (taken over all the jobs). That is, if we let $z(\sigma) = \max_j\{\alpha_j e_j(\sigma) + \beta_j t_j(\sigma)\}$, the objective is to find σ^* such that $z(\sigma^*) = \min_\sigma\{z(\sigma)\}$. Notice that, in this situation, there is no advantage to having inserted idle time between jobs. If d is sufficiently large, there may be an advantage in having inserted idle time before the first job. Without losing much, we assume that this is not the case. (Restricting the due-date not to be very large, as we do, is not a serious limitation and may actually lead to a more difficult problem.) Thus, a potentially optimal schedule starts at time 0 and has the completion time of every job equal the start time of its immediate successor. This implies that a schedule can be completely characterized by the associated job sequence (which is simply an ordered set of the job indices). Before moving on, we assume that all job parameters (p_j, α_j, β_j and d) are integers and further that $\alpha_j = \beta_j$ (earliness and tardiness penalties are symmetric).

That segment of the machine scheduling literature which involves earliness and tardiness penalties, while relatively new, is rather vast. It is not our intention to review that literature thoroughly. We choose instead to focus only on those works that are most closely related to ours. An interested reader may refer to the surveys by Baker and Scudder [1] and Gordon et al. [4] for the missing details. Turning to the problem at hand, it receives full treatment for the first time at the hands of Li and Cheng [5], who show that the problem is NP-hard [3] for a single machine (and strongly NP-hard [3] for an arbitrary number of identical parallel machines) and go on to provide an approximate solution with a performance guarantee. Cheng et al. [2] handle the multiple machine version of the problem and give a genetic-algorithm for its approximate solution. Mosheiov and his associates work extensively on the special case where $p_j = 1$ (the jobs are unit execution time or UET jobs). Mosheiov and Shadmon [7] treat both the single and multiple machine versions of the problem under the UET assumption and provide polynomial-time (formally efficient) algorithms in most cases; when such an algorithm is not given, they give an approximate solution with a performance guarantee. Mosheiov [6] extends the UET work to the case of a flowshop and again gives a polynomial time solution algorithm. Most recently, Mosheiov and Yovel [8] frame the problem in a due-date assignment context and obtain similar results.

As noted, the single machine problem that is of interest to us here is in general hard. But it is not clear how hard it really is. It does not have a pseudo-polynomial time [3] solution yet. Thus, while we know that it is NP-hard, we do not know if it is so in the ordinary sense or in the strong sense. In this paper, we first consider a special case of the problem (which in fact subsumes the UET problem as a sub-case) and show that this case (which remains NP-hard) admits a pseudo-polynomial-time (practically efficient) solution (which also solves the UET sub-case in polynomial time). We then hint how this algorithm can be made to yield a fully polynomial time approximation scheme and extend to the multiple machine case. We next turn to the general problem and present a heuristic algorithm, which extends easily to the asymmetric weight ($\alpha_j \neq \beta_j$) case as well. The heuristic, which is greedy and polynomial time, remains to be tested empirically before claims can be made about its performance.

2. Pseudo-Polynomial Time Solvable Special Case

Recall that our primary concern is a single machine scheduling problem with a MINMAX objective, symmetric, job-specific earliness and tardiness penalties, and a common due-date that is not too large. Extending standard notation, the problem can be denoted by $1/d_j=d,\alpha_j=\beta_j/\max_j\{\alpha_j e_j + \beta_j t_j\}$. We assume that d is small (small enough for there to be an optimal schedule with a start time of 0), just for the simplicity of exposition; a large d can easily be handled through a minor modification of our solution algorithm. Also for convenience, we assume that the jobs are numbered such that $\alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_n$. (In case $\alpha_i = \alpha_j$, we make $i < j$ if $\alpha_i/p_i \geq \alpha_j/p_j$.) Note that our choosing to work with α_j (rather than β_j) is purely idiosyncratic.

2.1 Problem and Solution Characteristics

We now state a few observations that hold for the general case and extend to the special case as well.

Observation 1: The general problem is NP-hard.

The proof appears in [5]. We will briefly outline it later.

Observation 2: In an optimal schedule for the general problem, if two jobs i and j appear next to each other:

(a) when both are early, i follows j if $\alpha_i \geq \alpha_j$ and $\alpha_i/p_i \geq \alpha_j/p_j$, and

(b) when both are tardy and start after d, i precedes j if $\alpha_i \geq \alpha_j$.

The proof is easily accomplished via an interchange argument.

The special case that we consider here assumes that $i < j \Rightarrow \alpha_i \geq \alpha_j$ and $\alpha_i/p_i \geq \alpha_j/p_j$ for all i and j, that is, the jobs can be numbered such that both conditions in Observation 2 are satisfied. (Notice that UET is a sub-case.) The upshot is that the early jobs in an optimal schedule are scheduled in decreasing order of their indices, whereas the tardy jobs (with the possible exception of the job that straddles d) are scheduled in increasing order of their indices. This is what makes it possible for us to develop a pseudo-polynomial time dynamic program to solve the special case.

It may be worthwhile at this point to note that our special case is NP-hard as well. The fact that we are able to solve it in pseudo-polynomial time (as shown later) establishes that it is ordinary NP-hard. The NP-hardness proof is very similar to that in [5]. Take n+1 jobs with: $\alpha_1 = A$, $\alpha_2 = \alpha_3 = \ldots = \alpha_n = \alpha_{n+1} = 1$; p_j for $j = 1,\ldots, n$ are as usual and add up to 2B, but $p_{n+1} = C$; and $d = B + C$. Here, A and C are appropriately large numbers. The question is: is there a schedule σ with $z(\sigma) \leq B$? Note first that the problem instance just created satisfies the special case criteria. Note next that, for the question posed above to have an affirmative answer: job 1 must complete at d, job n+1 must start at 0 and jobs 1 through n must have a partition such that the processing times of the jobs on each side of the partition add up to B. This essentially provides a reduction from the well-known NP-hard problem called Partition [3].

2.2. Pseudo-Polynomial Time Dynamic Program

We have noted that an optimal schedule for the special case has a particular structure. Jobs finishing before d appear in decreasing order of their indices and those starting after d in increasing order. But the identity of the job that straddles d (call it the pivot job) remains unclear; so we try all n jobs, one at a time. Say q is the pivot job in one such try. We build partial schedules outside in by considering the remaining n-1 jobs in decreasing order of their indices and by placing them, when their turn comes, on the inside left or the inside right of the schedule. When all jobs but job q has been scheduled, we fill the void left by placing q there.

Suppose that, when working with a particular pivot job q, the remaining n-1 jobs have been re-indexed from 1 to n-1 (retaining their original order) and job q has been re-indexed 0. At stage k of the dynamic program when job n-k has just been scheduled, let $f_{q,k}(x)$ be the minimum of the maximum earliness-tardiness penalty of all k-job partial schedules whose early set completes processing at time x. Clearly, the partial schedule that yields this minimum dominates others for the given x and is retained for further expansion. Let $\delta_{q,k}(x)$ take on the value 1 if the retained partial schedule (for the given x) has job n-k in the early set and 0 otherwise. Finally, let P be the total processing time of all the n jobs and P_k the total processing time of the k higher indexed jobs considered up to stage k in this run of the dynamic program corresponding to the chosen pivot job q.

The dynamic programming recursions can be stated now.

For k = 0:

$$f_{q,0}(x) = 0 \text{ for } x = 0, \text{ and } = \infty \text{ otherwise.}$$

For k = 1 to n-1:

$$f_{q,k}(x) = \min\{\max\{f_{q,k-1}(x-p_{n-k}), \alpha_{n-k}(d-x)\}, \max\{f_{q,k-1}(x), \alpha_{n-k}(P-P_{k-1}+x-d)\}\} \quad \text{for } \max\{0, d-P+P_k\} \le x \le d, \text{ and } = \infty \text{ otherwise.}$$

$\delta_{q,k}(x)$ is computed along with $f_{q,k}(x)$ so that an optimal schedule can be constructed through backtracking at the very end. That happens when job q is finally factored in and the optimal solution value corresponding to q as the pivot job is calculated as follows: $f_q^* = \min_x\{\max\{f_{q,n-1}(x), \alpha_0(x+p_0-d)\}\}$.

Once we have run the dynamic program n times (once for each possible value for q), we have n solutions. We pick the best as: $f^* = \min_q\{f_q^*\}$. This solves the special case. The dynamic program implicitly considers all partial schedules that can lead to optimality and retains only the non-dominated ones. It is thus correct. As for the computation: at the end of each stage of the dynamic program, at most d partial schedules are retained; there are n stages in a given run of the dynamic program; and, there are n runs. This indicates that the dynamic program takes $O(n^2d)$ time.

Note that, if we apply our method to the UET sub-case, we get an $O(n^2)$ time solution (p_j being 1, there is no need to consider the n possibilities for the pivot job). If we let p_j equal p for all j (in a somewhat generalized version of UET), we get an $O(n^3)$ time solution. A bottleneck assignment problem formulation in the latter case yields a solution of the same time order. However, Mosheiov and Shadman [7] provide a much better O(nlogn) time solution for the former.

2.3. Possible Extensions

We have said at the outset that we will restrict ourselves to a due-date that is not large and enforce a 0 start time on the schedule. From a technical point of view, this is not really necessary. (We do this only to make our exposition easy to follow.) The dynamic programming solution given above can easily accommodate a non-zero start time. We simply have to let d be arbitrary and modify the initialization step (stage 0) so that $f_{q,0}(x) = 0$ for $0 \le x \le d$.

We have also said that it is possible to develop, for the special case, a fully polynomial time approximation scheme (FPTAS) based on the dynamic program. The details are rather technical and we omit them here. We only note that an FPTAS, in our case, will deliver a solution whose value is within $(1+\varepsilon)$ times, $\varepsilon > 0$, the optimal solution value and will run in time that is polynomial in n, $\log(\alpha_j)$, $\log(p_j)$, log(d) and $(1/\varepsilon)$ for all j. It does this by retaining a subset of the $f_{q,k}(x)$ values at stage k, for example, by keeping only the minimum $f_{q,k}(x)$ value for all the x values belonging to a given interval of width Δ. The challenge essentially is in the choice of an appropriate Δ. We can report that such a Δ exists in this instance.

While we have said that our methods extend to the multiple machine case, the modifications required are not trivial. In this case, one needs to develop a dynamic program such that the partial schedules are built inside out around the pivot job. This approach handles the non-zero start times more naturally and can be used to solve the single machine problem as well. Its difficulty basically lies in the fact that it is not easy to describe. Finally, we should note that the multiple machine version of the special case is NP-hard as well; in fact, it is strongly NP-hard for an arbitrary number of machines m. The time complexity of the dynamic program that we propose grows exponentially with m.

3. Heuristic Solution for the General Problem

Our heuristic is greedy and polynomial time; in fact, it runs in $O(n^2)$ time. It applies to both the symmetric and asymmetric penalty versions of the problem, even though we describe it here for the former. It works with fixed schedule start and finish times and does not extend naturally to the multiple machine version. We also have to assume that d is not large or trivial, viz., that $\max_j\{p_j\} < d < P$.

At any stage during schedule construction, let S and U be, respectively, the set of jobs that have already been scheduled and that remain to be scheduled. Similarly, let L and R be, respectively, the set of jobs that have been scheduled leftmost and rightmost in S; thus, $S = \{L| R\}$. Suppose that the jobs in L start at time 0 and complete at time x; likewise, suppose that the jobs in R complete at time P and start at time y. It is easy to see at this juncture that there is a hole in the schedule between times x and y, where the jobs in U will have be scheduled. For convenience, let us define the following terms: z_j^L = earliness penalty of job j if it is scheduled last in L; z_j^R =tardiness penalty of job j if it is scheduled first in R; z_j = minimum of z_j^L and z_j^R; δ_j = indicator of whether z_j equals z_j^L or z_j^R; and, z^* = the maximum earliness-tardiness penalty of the jobs in S.

The algorithmic details can now be given as follows.

```
Step 0: Let: $x = 0$, $y = P$, $z^* = 0$, $L = \emptyset$, $R = \emptyset$, $S = \emptyset$ and $U = \{1, 2, \ldots, n\}$.

Step 1: Do until U is empty:

    a.  For each $j \in U$, do:
            If $x + p_j > d$: $z_j^L = \infty$; else, $z_j^L = \alpha_j (d - x - p_j)$.
            If $y \leq d$: $z_j^R = \infty$; else $z_j^R = \alpha_j (y - d)$.
            If $z_j^L < z_j^R$: $z_j = z_j^L$ and $\delta_j = 1$; else, $z_j = z_j^R$ and $\delta_j = 0$.

    b.  Find $j^*$ such that $z_{j^*} = \min_{j\in U}\{z_j\}$.
        If $\delta_{j^*} = 1$: $L = \{L| j^*\}$ and $x = x + p_{j^*}$; else, $R = \{j^*| R\}$ and $y = y - p_{j^*}$.
        $S = \{L| R\}$, $U = \{U - j^*\}$ and $z^* = \max\{z^*, z_{j^*}\}$.

Step 2: Deliver S as the heuristic schedule with minmax penalty equal to $z^*$.
```

Notice that the loop in step 1 executes n times and that at each execution O(n) computations are involved. The overall time complexity of the heuristic is thus $O(n^2)$.

4. Conclusions

In this paper, we have addressed a single machine scheduling problem whose objective is to minimize the maximum of the earliness and tardiness penalties about a common due-date. We have focused on the version where the earliness and tardiness penalties are the same for the same job but different for different jobs. This problem is known to be computationally hard.

For an important special case which remains hard, we have exploited certain structural properties of an optimal schedule to develop a pseudo-polynomial time dynamic programming algorithm. We have shown how this algorithm can be made to accommodate non-zero start times (which arise when the due-date is sufficiently large). We have also discussed, albeit briefly, how this algorithm can be used to develop a fully polynomial time approximation algorithm. Furthermore, we have indicated, again briefly, how the multiple machine version of the special case can be solved exactly using a dynamic program similar to ours.

For the general problem, we have proposed a polynomial time greedy heuristic. While this heuristic is intuitively appealing, its performance is yet to be tested. The good thing about the heuristic is that it applies, with little modification, to the generalization where the earliness and tardiness penalties are allowed to be asymmetric (different even for the same job). Its weakness is its reliance on the schedule starting at time 0.

In terms of future work, the performance (quality-wise) of the heuristic needs to be tested. It will also be nice to see if a performance guarantee can be established for it.

References

[1] K. Baker and G. Scudder, Sequencing with Earliness and Tardiness Penalties: A Review, Operations Research, Vol. 38, pp. 22-36, 1990.

[2] R. Cheng, M. Gen and T. Tozawa, Minmax Earliness/Tardiness Scheduling in Identical Parallel Machines Using Genetic Algorithms, Computers & Industrial Engineering, Vol. 29, pp. 513-517, 1995.

[3] M. Garey and D. Johnson, Computers and Intractability, W. H. Freeman and Company, San Francisco, 1979.

[4] V. Gordon, J. Proth and C. Chu, A Survey of the State-of-the-Art of Common Due date Assignment and Scheduling Research, European Journal of Operational Research, Vol. 139, pp.1-25, 2002.

[5] C.-L. Li and T. Cheng, The Parallel Machine Min-Max Weighted Absolute Lateness Scheduling Problem, Naval Research Logistics, Vol. 41, pp. 33-46, 1994.

[6] G. Mosheiov, Scheduling Unit Processing Time Jobs on an m-Machine Flow-Shop, Journal of the Operational research Society, Vol. 54, pp.437-441, 2003.

[7] G. Mosheiov and M. Shadmon, Minmax Earliness-Tardiness Costs with Unit Processing Time Jobs, European Journal of Operational Research, Vol. 130, pp.638-652, 2001.

[8] G. Mosheiov and U. Yovel, Minimizing Weighted Earliness-Tardiness and Due-Date Cost with Unit Processing-Time Jobs, European Journal of Operational Research, Vol. 172, pp.528-544, 2006.