A note on polynomially-solvable cases of common due date early-tardy scheduling with release dates

Jorge M. S. Valente * Rui A. F. S. Alves *

* Faculdade de Economia, Universidade do Porto
{jvalente, ralves}@fep.up.pt

Abstract

In this paper we consider the single machine scheduling problem with integer release dates and the objective of minimising the sum of deviations of jobs' completion times from a common integer due date. We present an efficient polynomial algorithm for the unit processing time case. We also show how to calculate in polynomial time the minimum non-restrictive due date for the general case.

Keywords: scheduling, early-tardy, common due date, unit processing times, release dates

1 Introduction

The scheduling problem considered in this paper can be stated as follows. A set of n independent jobs $\{J_1, J_2, \cdots, J_n\}$, each with a possibly different integer release date r_j, a common integer due date d and a processing time p_j, has to be scheduled without preemptions on a single machine that can handle at most one job at a time. The objective is to minimise the sum of the absolute deviations of the jobs' completion times from the common due date $\sum_{j=1}^{n} |C_j - d|$, where C_j is the completion time of job J_j. In the classification scheme proposed by Lawler, Lenstra, Rinnooy Kan and Shmoys [6], this problem can be represented as $1\,|d_j = d, r_j|\sum |C_j - d|$. Scheduling models with both earliness and tardiness costs are particularly appealing, since they are compatible with the philosophy of just-in-time production. The model is also made more realistic by the existence of different release dates, since in most real production settings the orders are released to the shop floor over time (and not all simultaneously). Therefore, the problem considered has several potential practical applications.

The identical release dates version of this problem with a non-restrictive due date (i.e., a due date that does not constrain the optimal schedule cost) - $1\,|d_j = d_{nr}|\sum |C_j - d|$ - can be solved in $O(n \log n)$ time by algorithms presented by Kanet [5] and Bagchi, Sullivan and

Chang [1]. Hall, Kubiak and Sethi [4] proved that the restrictive version $1\,|d_j = d_r|\sum |C_j - d|$ is $NP-hard$. Models with different release dates have been addressed by Nandkeolyar, Ahmed and Sundararaghavan [7], Sridharan and Zhou [8] and Bank and Werner [3]. Nandkeolyar, Ahmed and Sundararaghavan [7] consider the single machine problem $1\,|r_j|\sum c_j\,|d_j - C_j|$, where c_j and d_j are, respectively, the cost per unit time and the due date of J_j. They present several heuristics, developed in a modular fashion. Sridharan and Zhou [8] present a decision theory based approach for a more general problem where the cost per unit time is allowed to be different according to whether the job is early or tardy. Bank and Werner [3] consider a model with unrelated parallel machines, a common due date and earliness and tardiness costs that may differ between jobs. They present several constructive and iterative heuristics. It should be pointed out that there are a large number of papers considering earliness and tardiness penalties. Only the papers above are reviewed because they consider problems that are closest to ours. For more information on problems with earliness and tardiness costs, the interested reader is referred to Baker and Scudder [2], who present a comprehensive survey of early/tardy scheduling.

In section 2 a $O(n \log n)$ algorithm for the problem with unit processing times

$$1\,|p_j = 1, d_j = d, r_j|\sum |C_j - d|$$

is presented. A polynomial algorithm for calculating the minimum non-restrictive value of d in the general case $1\,|d_j = d, r_j|\sum |C_j - d|$ is also given in section 3.

2 An algorithm for problem $1\,|p_j = 1, d_j = d, r_j|\sum |C_j - d|$

In this section a $O(n \log n)$ algorithm for the problem with unit processing times is presented. Several lemmas and theorems are first developed. These lemmas and theorems characterize the structure of an optimal solution. The algorithm then simply schedules the jobs in such a way that the lemmas and theorems are satisfied (hence optimally).

Lemma 1 *There exists an optimal sequence where each C_j is integer.*

Proof. Any feasible schedule with non integer C_j's can be transformed into a feasible sequence, of lower or equal cost, where all C_j's are integer. Starting from d and scanning left, take the first job, if any, with a non integer $C_j < d$ such that there exists idle time to the right of that job. Move that job to the right until it is blocked by another job or it completes at an integer time, whichever occurs first. Any such movement will decrease the cost of the schedule. Repeat until no such jobs exist. Perform a similar scan to the right of d, this time moving jobs to the left (such a move is always feasible because each job has an integer r_j and we stop as soon as an integer start time is reached, if the job is not blocked before). Again, any such movement decreases the schedule cost. At this time, at most one group of jobs performed consecutively with no idle time in between (or possibly just a single job) starts and completes at non integer times that encompass d. Let A and B denote the sets of jobs in that group that complete after and before d, respectively. If $|A| > |B|$, move the block backwards until an integer start time is reached. The schedule cost will decrease, since the earliness of each

job in B increases by the amount of backwards movement but the tardiness of each job in A decreases by that same amount. If $|A| < |B|$, move the block forward instead. If $|A| = |B|$, move either forward or backward. The new schedule has all integer C_j and its cost is lower or the same. ■

This lemma allows us to focus on unit time slots that begin at integer times, since there exists an optimal sequence where jobs are scheduled in n of those slots. This result also indicates that the problem could be formulated as a weighted bipartite matching problem, and therefore solved in $O\left(n^3\right)$ time, but a more efficient approach is possible. The next lemma identifies the best possible schedule.

Lemma 2 *Any feasible sequence in which the jobs are scheduled in the n consecutive time slots in the time range* $\left[d - \left\lceil \frac{n}{2} \right\rceil, d + \left\lfloor \frac{n}{2} \right\rfloor\right]$ *is also optimal.*

Proof. No slot in this range has a cost higher than $\left\lfloor \frac{n}{2} \right\rfloor$. Since any slot not in this time range has a cost that is at least as high as this value, the jobs are indeed scheduled in the n least cost slots and the sequence is therefore optimal. ■

Assume, for the remainder of this section, that jobs have been renumbered in non decreasing order of their release dates. Let EC_j be the earliest possible completion time of job j when jobs are considered for processing in increasing index order. Let $EC = EC_n$ denote the earliest possible completion time of the job with the largest index. Also note that, according to their definition, all EC_j's must be different.

Lemma 3 *It is possible to schedule the jobs in the n consecutive time slots in the time range* $\left[d - \left\lceil \frac{n}{2} \right\rceil, d + \left\lfloor \frac{n}{2} \right\rfloor\right]$ *if and only if* $EC \leq d + \left\lfloor \frac{n}{2} \right\rfloor$.

Proof. If $EC > d + \left\lfloor \frac{n}{2} \right\rfloor$ then obviously at least one job cannot be completed up to $d + \left\lfloor \frac{n}{2} \right\rfloor$. If $EC \leq d + \left\lfloor \frac{n}{2} \right\rfloor$, any job with $EC_j > d - \left\lceil \frac{n}{2} \right\rceil$ can be scheduled to complete at its EC_j while the remaining jobs can be arbitrarily assigned to the still empty time slots in the optimal range. That assignment is clearly feasible, since the start time of each of those jobs is being delayed. ■

The next theorem identifies the minimum non-restrictive due date.

Theorem 4 *The due date d is non-restrictive when* $d \geq EC - \left\lfloor \frac{n}{2} \right\rfloor$.

Proof. Lemma 2 identifies the best possible schedule. From lemma 3, that schedule is feasible only if $d \geq EC - \left\lfloor \frac{n}{2} \right\rfloor$. ■

The next lemmas provide further characteristics of an optimal solution that will be used in the algorithm.

Lemma 5 *If* $EC > d + \left\lfloor \frac{n}{2} \right\rfloor$, *all jobs will be scheduled in the time range* $\left[d - \left\lceil \frac{n}{2} \right\rceil, EC\right]$ *in an optimal sequence.*

Proof. It is clear that all slots in the range have a lower cost than any slot that starts at or later than EC. Also jobs with $EC_j > d - \lceil \frac{n}{2} \rceil$ can once again be scheduled to complete at their EC_j while the remaining jobs can be arbitrarily assigned to the still empty time slots in the range $\left[d - \lceil \frac{n}{2} \rceil, d + \lfloor \frac{n}{2} \rfloor\right]$, thereby decreasing their cost. ■

Lemma 6 *There exists an optimal schedule in which all jobs with $EC_j \geq d$ are scheduled to complete at their EC_j.*

Proof. Consider the slots with a completion time equal to EC_j, for $EC_j \geq d$. If a job with $EC_j \geq d$ is completed later than its EC_j, and the slot with a completion time equal to EC_j is free, one can simply move that job into this slot (thereby decreasing its cost, since it will be completed earlier). Also, any feasible schedule in which any slot with a completion time equal to EC_j, for $EC_j \geq d$, is occupied by a job with $EC_j < d$ (an offending job) can be converted into an equal cost sequence in which all such slots are filled with jobs with $EC_j \geq d$. For each of those slots, any offending job is simply swapped with the job whose EC_j is equal to that slot's completion time. These swaps do not change the schedule cost and feasibility is maintained (the release date of the offending job is not larger than that of the job with which it is swapped, since its EC_j is lower, so it can also be feasibly scheduled in its destination slot). After all such swaps are performed, only jobs with $EC_j \geq d$ use those slots, though they are not necessarily in EC_j order (jobs are considered in increasing order of their indexes when calculating the EC_j's, so some jobs can feasibly be scheduled before their EC_j). If that is the case, reordering the jobs according to their EC_j will not alter cost nor feasibility. Therefore, all jobs with $EC_j \geq d$ can be optimally scheduled to complete at their EC_j. ■

Lemma 6 assigns optimal slots for jobs with $EC_j \geq d$, so all that remains is to optimally schedule the remaining jobs in the available slots. The next lemma considers those jobs.

Lemma 7 *Given that jobs with $EC_j \geq d$ are scheduled according to lemma 6, the remaining jobs should be scheduled in the available slots that are closest to d (i.e., the lowest cost slots).*

Proof. It simply needs to be proved that such a schedule is feasible, since it is clearly optimal. Let $|B|$ be the number of jobs with $EC_j < d$ (and therefore of necessary slots). The earliest possible completion time of the available slots that are closest to d is $d-1, d-2, \cdots, d-|B|$. The latest possible EC_j's of the remaining jobs are also $d-1, d-2, \cdots, d-|B|$, so they can be feasibly assigned to the least cost slots. Since any other case involves later slots and/or earlier EC_j's, it is always feasible to schedule the remaining jobs in the least cost available slots. An easy way for an algorithm to ensure feasibility is to simply consider jobs in decreasing order of their EC_j. ■

An algorithm that schedules jobs in such a way that the previous lemmas are satisfied is now presented. The algorithm uses a min heap of free time slot ranges and their associated minimum cost (the cost of the best slot in that range), which serves as the key for pushing and popping elements from the heap.

Algorithm 1

Step 1: Sort and renumber jobs in non decreasing order of r_j.

Step 2: Calculate EC_j for all jobs.

Step 3: If $EC < d + \lfloor \frac{n}{2} \rfloor$, push range $\left[EC, d + \lfloor \frac{n}{2} \rfloor\right]$ on heap.

Step 4: For each job, in decreasing order of EC_j, do:

If $EC_j \geq d$

schedule j to complete at its EC_j;
if $j > 1$, push range $[EC_{j-1}, EC_j - 1]$ on heap;

Else

schedule j to complete at its EC_j or at the best available free time slot on the heap (whichever has a lower cost; ties can be broken arbitrarily); in the latter case update the range that included that slot and re-insert it on heap;
if $j > 1$, push on heap:
range $[EC_{j-1}, EC_j - 1]$ if j completes at its EC_j;
range $[EC_{j-1}, EC_j]$ if j was scheduled at the best available free time slot on the heap.

In the previous algorithm ranges are obviously only pushed on the heap when the upper limit is higher than the lower limit. Updating a time range that ends at or before d or begins at or after d simply involves increasing its minimum cost by one and decreasing its finish time, or increasing its start time, respectively, by one time unit (thereby eliminating its previously best slot). Only one range that contains d as an interior point can be generated. When such a range is updated, it's divided into the two separate ranges that result from eliminating the time slot which finishes at d. Step 1 takes $O(n \log n)$ time and Step 2 $O(n)$ time. Step 3 can be done in constant time. In Step 4, the For loop is executed n times. In each iteration pushing or popping the heap takes $O(\log n)$ time and scheduling the job and updating time ranges (when necessary) takes $O(1)$ time. Therefore, the complexity of the algorithm is $O(n \log n)$.

Theorem 8 *Algorithm 1 generates an optimal schedule.*

Proof. The theorem follows from the previous lemmas. Jobs with $EC_j \geq d$ are scheduled as in lemma 6. The algorithm pushes all available time slots with completion time not later than $\max\left(EC, d + \lfloor \frac{n}{2} \rfloor\right)$ on the heap and jobs with $EC_j < d$ are scheduled on the best of those slots, as established in lemma 7. Note that when $EC \leq d + \lfloor \frac{n}{2} \rfloor$, jobs with $EC_j \geq d$ are scheduled to finish inside the optimal range $d - \lceil \frac{n}{2} \rceil$ to $d + \lfloor \frac{n}{2} \rfloor$. The remaining jobs will also be scheduled inside this range, since the algorithm will push its slots into the heap (note that range $\left[EC, d + \lfloor \frac{n}{2} \rfloor\right]$ is pushed on the heap). ■

In table 1 an example for Algorithm 1 is presented; assume $d = 7$. The jobs are already renumbered in non decreasing order of r_j. In step 2 the following EC_j's are calculated: $EC_1 = 1$; $EC_2 = 3$; $EC_3 = 4$; $EC_4 = 6$ and $EC_5 = 8$. The due date is in this case non-restrictive, since $EC \leq d + \lfloor \frac{n}{2} \rfloor$ $(8 \leq 7 + 2)$. In step 3 the range $[8, 9]$ (cost: 2) is pushed on

Table 1: Algorithm 1 example.

index	1	2	3	4	5
r_j	0	2	2	5	7

the heap. In step 4 the jobs are scheduled in decreasing index order. Since $EC_5 \geq d$, job 5 is scheduled in the slot $[7, 8]$ and range $[6, 7]$ (cost: 0) is pushed on the heap. Job 4 is then considered, and $EC_4 < d$. If job 4 is scheduled to complete at $EC_4 = 6$ its cost will be 1; if it is scheduled in the best slot available on the heap, its cost is 0. Therefore, job 4 is scheduled in the slot $[6, 7]$ and range $[4, 6]$ (cost: 1) is pushed on the heap. Job 3 is the next job to be scheduled, and $EC_3 < d$. If job 3 is scheduled to complete at $EC_4 = 4$ its cost will be 3; if it is scheduled in the best slot available on the heap (slot $[5, 6]$), its cost is 1. Job 3 is then scheduled in the slot $[5, 6]$, and both the updated range $[4, 5]$ (cost:2) and the new range $[3, 4]$ (cost: 3) are inserted in the heap. The remaining jobs will then be scheduled in the slots $[8, 9]$ and $[4, 5]$. The algorithm schedules the jobs in the optimal range $\left[d - \left\lceil \frac{n}{2} \right\rceil, d + \left\lfloor \frac{n}{2} \right\rfloor\right]$, in this case $[4, 9]$.

3 Calculating the minimum non-restrictive common due date for the general case $1\,|d_j = d, r_j|\sum |C_j - d|$

With different release dates, the due date d will be non-restrictive when the optimal schedule for the non-restrictive version of the problem with equal release dates is feasible, since clearly no better schedule can be generated. Therefore, the minimum value of the common due date d for which that schedule is feasible must be found. Throughout this section assume that the jobs have been renumbered in non decreasing order of p_j. An optimal schedule for the non-restrictive version of the problem with equal release dates can be determined by the following procedure presented by Kanet [5]. Let B be a sequence of jobs to be scheduled without idle time such that the last job in B is completed at d. Let A be a sequence of jobs to be scheduled without idle time such that the first job in A starts at d. An optimal schedule for the problem with identical release dates consists of B followed by A, given that those sequences are generated by the following rule: assign jobs alternately, and in their index order, to the beginning of B and end of A, starting with B if n is odd and A otherwise. The minimum non-restrictive common due date for the problem with equal release dates, which will be denoted as $\Delta_{r=0}$, is then $\Delta_{r=0} = \sum_{j \in B} p_j$.

The minimum non-restrictive due date when different release dates are allowed will now be considered. When all jobs share a common release date r, the smallest non-restrictive common due date will simply be $\Delta_{r=0} + r$. If jobs have different release dates as well as different processing times, one can determine the start time of each job in the schedule generated by Kanet's procedure (assuming all release dates equal to zero) and calculate the maximum violation of a release date (i.e., the maximum positive difference between a job's release date and its start time in the Kanet schedule). The minimum non-restrictive common due date could then be obtained by adding that maximum violation to $\Delta_{r=0}$. When different jobs have identical processing times, the situation is more complicated. Since processing times are not unique, ties occur when renumbering the jobs in non decreasing order of p_j, and several different Kanet schedules may be generated, each leading to a possibly different maximum

violation of a release date. Therefore, when renumbering jobs, ties must be broken in such a way that the resulting Kanet schedule minimizes the maximum violation of a release date. The following algorithm generates the minimum non-restrictive value of the common due date d (denoted by Δ) when release dates are allowed to be different, and different jobs may have identical processing times. If all release dates are identical, the algorithm is equivalent to Kanet's procedure. Let p_A and p_B denote, respectively, the sum of the processing times of the jobs currently assigned to A and B. Jobs are added to the beginning of B and to the end of A and the first job is to be assigned to B (A) if the number of jobs n is odd (even).

Algorithm 2

Step 1: Sort and renumber jobs in non decreasing order of p_j; break ties by choosing job with lower r_j.

Step 2: Set p_A, p_B and Δ to 0.

Step 3: Consider jobs in increasing index order:

If p_j is unique

If j is the first job to be assigned, assign j to B (A) if n is odd (even); otherwise assign j to B (A) if last job was assigned to A (B).

If j is added to B

let $\Delta_j = r_j + p_j + p_B$;

If $\Delta_j > \Delta$, set $\Delta = \Delta_j$;

$p_B = p_B + p_j$;

Else

let $\Delta_j = r_j - p_A$;

If $\Delta_j > \Delta$, set $\Delta = \Delta_j$;

$p_A = p_A + p_j$;

Else

let c be the number of jobs with that p_j;

$\left\lceil \frac{c}{2} \right\rceil$ jobs are assigned to A (B) and $\left\lfloor \frac{c}{2} \right\rfloor$ to B (A) if last job was assigned to B (A);

the jobs assigned to B are those with lower r_j, and are assigned in non increasing order of r_j;

the jobs assigned to A are those with higher r_j, and are assigned in non decreasing order of r_j;

update Δ_j, p_A and p_B as above.

The complexity of the algorithm is $O(n \log n)$, since Step 1 takes $O(n \log n)$ time, Step 2 takes constant time and Step 3 requires $O(n)$ time.

Theorem 9 *Algorithm 2 generates the minimum non-restrictive value for d.*

Proof. The algorithm clearly generates a sequence that is optimal for the problem with equal r_j. Any $d < \Delta$ leads to an infeasible schedule, since at least one job will not be available at its

Table 2: Algorithm 2 example.

index	1	2	3	4	5
p_j	5	7	7	8	10
r_j	0	6	8	7	5

optimal start time. However, when several jobs have identical p_j, several optimum sequences exist for the problem with equal r_j (those jobs will have to go into certain positions, but several assignments are possible). When this happens, the algorithm assigns the jobs with lower r_j to the earlier slots. Therefore, it needs to be shown that any other assignment does not lead to a lower Δ. Take any pair of jobs i and j such that $r_i < r_j$ but j is scheduled before i. Assume that j is in B and i is in A. When Δ is being calculated, we have $\Delta_j^1 = r_j + p_j + p_B$ and $\Delta_i^1 = r_i - p_A$, and $\Delta_j^1 > \Delta_i^1$. If those jobs were swapped, we would have $\Delta_i^2 = r_i + p_i + p_B$ and $\Delta_j^2 = r_j - p_A$. Since $\Delta_i^2 < \Delta_j^1$ and $\Delta_j^2 < \Delta_j^1$, the value of Δ cannot be higher after the swap. A similar reasoning applies when both jobs are in the same set. So swapping jobs until they are in r_j order will not increase Δ. ■

In table 2 an example for Algorithm 2 is presented. Jobs have already been renumbered in non decreasing order of p_j, with ties broken by lower r_j. In step 3 the jobs are considered in increasing index order. The first job's p_j is unique, and job 1 is assigned to B since n is odd. The algorithm then calculates $\Delta_1 = 0 + 5 + 0 = 5$. Since $\Delta_1 > \Delta = 0$, the algorithm sets $\Delta = \Delta_1 = 5$ and then updates $p_B = 5$. Jobs 2 and 3 have identical processing times. Job 3 (with the higher r_j) is then assigned to A, while job 2 (with the lower r_j) is assigned to B. The algorithm calculates $\Delta_3 = 8 - 0 = 8$, and sets $\Delta = \Delta_3 = 8$ and $p_A = 7$. When job 2 is assigned to B, $\Delta_2 = 6 + 7 + 5 = 18$. Since $\Delta_2 > \Delta = 8$, the algorithm sets $\Delta = \Delta_2 = 18$ and then updates $p_B = 5 + 7 = 12$. The processing time of job 4 is unique, and this job is assigned to A. Since $\Delta_4 = 7 - 7 = 0$, Δ is not changed, while p_A is updated to 15 (7+8). Finally, job 5 is assigned to B and $\Delta_5 = 5 + 10 + 12 = 27$. Therefore, Δ is set at 27, which is the minimum non-restrictive due date.

Acknowledgement

The authors would like to thank an anonymous referee for several helpful comments that were used to improve this paper.

References

[1] Bagchi, U., Sullivan, R., and Chang, Y. Minimizing mean absolute deviation of completion times about a common due date. *Naval Research Logistics Quarterly 33* (1986), 227–240.

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

[3] Bank, J., and Werner, F. Heuristic algorithms for unrelated parallel machine scheduling with a common due date, release dates, and linear earliness and tardiness penalties. *Mathematical and Computer Modelling 33* (2001), 363–383.

[4] Hall, N., Kubiak, W., and Sethi, S. Earliness-tardiness scheduling problems, ii: Deviation of completion times about a restrictive common due date. *Operations Research 39* (1991), 847–856.

[5] Kanet, J. Minimizing the average deviation of job completion times about a common due date. *Naval Research Logistics Quarterly 28* (1981), 643–651.

[6] Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., and Shmoys, D. B. Sequencing and scheduling: Algorithms and complexity. In *Logistics of Production and Inventory, Handbooks in Operations Research and Management Science, pp. 445-522*, S. C. Graves, A. H. G. Rinnooy Kan, and P. H. Zipkin, Eds. North-Holland, Amsterdam, 1993.

[7] Nandkeolyar, U., Ahmed, M. U., and Sundararaghavan, P. S. Dynamic single-machine-weighted absolute deviation problem: Predictive heuristics and evaluation. *International Journal of Production Research 31* (1993), 1453–1466.

[8] Sridharan, V., and Zhou, Z. A decision theory based scheduling procedure for single machine weighted earliness and tardiness problem. *European Journal of Operational Research 94* (1996), 292–301.