An Efficient Algorithm for Minimizing Earliness, Tardiness, and Due-Date Costs for Equal-Sized Jobs

Chung-Lun Li[1] Gur Mosheiov[2] Uri Yovel[3]

[1]Department of Logistics, The Hong Kong Polytechnic University,
Hung Hom, Kowloon, Hong Kong, China
lgtclli@polyu.edu.hk

[2]School of Business Administration and Department of Statistics,
The Hebrew University, Jerusalem 91905, Israel
msomer@mscc.huji.ac.il

[3]Graduate School of Business,
Columbia University, New York, NY 10027, U.S.A.
uy2102@columbia.edu

TECHNICAL NOTE

October 2006
Revised January 2007
Revised March 2007

Abstract

We consider a single-machine scheduling problem with equal-sized jobs. The objective is to minimize the maximum weighted earliness-tardiness and due-date costs. We present an algorithm to solve this problem. Our algorithm makes use of bottleneck jobs and priority queues, and has a computational complexity of $O(n^4 \log n)$. This complexity is a significant improvement of the existing algorithm in the literature.

Keywords: Analysis of algorithms; due-date assignment; earliness-tardiness; scheduling

1 Introduction

We consider a scheduling problem with n jobs, denoted $J_1, J_2, \ldots, J_n$, to be processed by a single machine. Each job J_j is characterized by its processing time $p_j > 0$, an earliness penalty $\alpha_j \geq 0$, and a tardiness penalty $\beta_j \geq 0$ ($j = 1, 2, \ldots, n$). All the jobs share a common due-date $d \geq 0$, which is to be determined. For a given job sequence, let C_j denote the completion time of J_j. Let $E_j = \max\{0, d - C_j\}$ and $T_j = \max\{0, C_j - d\}$ denote the job earliness and tardiness, respectively. There is a unit cost $\gamma \geq 0$ associated with (delaying) the due-date. We would like to obtain the schedule and the due-date that minimize the maximum cost incurred by any of the jobs. In other words, our objective is to minimize $\max_{j=1,\ldots,n}\{\alpha_j E_j + \beta_j T_j + \gamma d\}$. A job J_j is said to be early if $E_j > 0$; tardy if $T_j > 0$; and on-time if $E_j = T_j = 0$. Throughout this note, we assume that $n \geq 2$.

When $\alpha_j = \beta_j$ and $\gamma = 0$, the objective becomes a minimization of the Maximal Weighted Absolute Lateness and the problem is denoted as MWAL (see Gordon *et al.* [3]). Li and Cheng [4] have shown that the MWAL problem is NP-hard. Solving the general problem with arbitrary values of α_j, β_j, and γ is even more challenging. Such a generalized version of MWAL is denoted as GMWAL. Mosheiov and Yovel [5] have studied a special case of the GMWAL problem in which all jobs have identical processing times (i.e., $p_1 = p_2 = \cdots = p_n$), and they have developed an $O(n^{6.5}(\log n)^{0.5})$ algorithm for it. In this note, we develop a new algorithm for this special case, which is denoted as GMWAL(UJ). This algorithm makes use of *bottleneck jobs* and *priority queues*, and has an improved running time of $O(n^4 \log n)$. For simplicity, we assume, without loss of generality, that $p_1 = p_2 = \cdots = p_n = 1$.

Due-date assignment problems reflect many real-life situations where the due-date is determined during sales negotiations. Firms clearly benefit from delaying due-dates, because it increases their production flexibility. The later the due-date, the higher is the probability that a given job can be completed on time. However, a late due-date decreases the firm's competitiveness, making other producers/suppliers more attractive. As a result, in many cases firms offer price reductions when the actual due-date is much later than the expected one. This setting was first introduced in the early 80's. Later, numerous extensions and special cases were studied, as reflected in many of the 130 references mentioned in the survey paper of Gordon *et al.* [3]. Our work, as mentioned earlier, focuses on the special case of identical jobs, which is applicable to situations where the scheduler has to arrange the processing of a batch of equal-sized jobs on a single machine and to quote a common due-date for these jobs to their customers. We consider a min-max objective, which applies to the case in which each job J_j has an earliness tolerance of A_j and a tardiness tolerance B_j, where A_j and B_j are inversely proportional to α_j and β_j, respectively. The scheduler attempts to fit each job completion time C_j into the time window $[d - A_j, d + B_j]$. Different jobs belong to different customers with different expectation of service, and therefore, α_j and β_j are customer-dependent. In a scheduling environment like this, it is not uncommon to have hundreds of jobs in the given batch. Thus, having an efficient method for solving such a problem is necessary to ensure smooth operations.

2 A New Algorithm for GMWAL(UJ)

We first discuss some properties of the optimal solution to GMWAL(UJ). It is easy to see that there exists an optimal solution with no machine idle time. In that optimal solution, the start time of every job must be an integer. (Note: The optimal due-date may not be integer valued.) Hence, in our algorithm we only consider solutions in which all jobs have integer start times. However, we allow machine idle time in the solution. The following lemma states another property of the optimal solution. This new property is critical to the development of our algorithm.

Lemma 1 *There exists an optimal solution to GMWAL(UJ) in which one of the following conditions holds: (i) There is no tardy job; (ii) $d = 0$; (iii) there exist an early (or on-time) job J_e and a tardy job J_t such that*

$$\alpha_e(d - C_e) = \beta_t(C_t - d) = R; \tag{1}$$

(iv) there exist tardy jobs J_{t_1} and J_{t_2} such that

$$\beta_{t_1}(C_{t_1} - d) = \beta_{t_2}(C_{t_2} - d) = R; \tag{2}$$

where $R = \max_{j=1,\ldots,n}\{\alpha_j E_j + \beta_j T_j\}$.

Proof Suppose, to the contrary, that all four conditions are violated in any optimal solution to the problem. Then, consider one such optimal solution with the smallest possible due-date d. Clearly, in this optimal solution, $R > 0$ (otherwise either (i), (iii), or (iv) holds) and $d > 0$ (otherwise (ii) holds). Also, R must be either equal to $\alpha_e E_e$ for some early job J_e or equal to $\beta_t T_t$ for some tardy job J_t.

Case 1: $R = \alpha_e E_e$ for some early job J_e. In this case, $R > \beta_j T_j$ for $j = 1, 2, \ldots, n$ (because condition (iii) is violated). Thus, keeping the existing job schedule but reducing the due-date slightly will form a new solution with a lower total cost. This is a contradiction.

Case 2: $R = \beta_t T_t$ for some tardy job J_t. In this case, $R > \alpha_j E_j$ for $j = 1, 2, \ldots, n$ (because condition (iii) is violated) and $R > \beta_j T_j$ for all $j \in \{1, 2, \ldots, n\} \setminus \{t\}$ (because condition (iv) is violated). If $\gamma < \beta_t$, then keeping the existing job schedule but increasing the due-date slightly will form a new solution with a lower total cost, which is a contradiction. If $\gamma > \beta_t$, then keeping the existing job schedule but decreasing the due-date slightly will form a new solution with a lower total cost, which is a contradiction. If $\gamma = \beta_t$, then keeping the existing job schedule but decreasing the due-date slightly will form a different solution with the same cost. Such a solution is also optimal but with a smaller due-date, which is again a contradiction. ∎

Based on Lemma 1, we propose the following algorithm: We first solve GMWAL(UJ) under condition (i), then solve it under condition (ii), and so on. We select the best one among the four solutions.

Condition (i) of Lemma 1 is satisfied only if $\alpha_j = 0$ for at least $n - 1$ jobs. (If $\alpha_j = 0$ for less than $n - 1$ jobs, then the earliness cost of the "no tardy job solution" must be positive. In such a case, decreasing the due-date slightly will form an alternate solution with a lower total cost, which contradicts the optimality of the solution.) Under condition (i), there exists an optimal schedule in which: (a) there is no machine idle time; (b) the job with $\alpha_j > 0$, if exists, is scheduled as the last job; and (c) the due-date is at $C_{\max} = n$. Thus, in this case, the problem can be solved in $O(n)$ time.

If condition (ii) of Lemma 1 is satisfied, then there exists an optimal schedule in which: (a) $d = 0$; (b) there is no machine idle time; and (c) all jobs are sequenced in nonincreasing order of β_j. Thus, in this case, the problem can be solved in $O(n \log n)$ time.

If the optimal schedule satisfies condition (iii), then there exist a "bottleneck early job" J_e and a "bottleneck tardy job" J_t such that both the earliness cost of J_e and the tardiness cost of J_t are equal to R. Now, we present a solution method for the GMWAL(UJ) problem under condition (iii). We select: (a) a bottleneck early job J_e; (b) a bottleneck tardy job J_t; and (c) the difference between the completion times of J_e and J_t, denoted as Y. We determine if this combination of J_e, J_t, and Y yields a "feasible" solution, that is, a solution with J_e and J_t being the bottleneck early and tardy jobs, respectively. If it yields a feasible solution, then we determine the smallest start time for J_e such that a feasible solution exists. We repeat this procedure with all possible combinations of J_e,

J_t, and Y. As mentioned earlier, there exists an optimal solution to GMWAL(UJ) with no machine idle time. Thus, it suffices to consider $Y = 1, 2, \ldots, n-1$. Among all feasible solutions generated, we select the one with the lowest total cost. This method can generate an optimal solution to the GMWAL(UJ) problem because for any given J_e, J_t, and Y, the best possible feasible solution is the one with the smallest due-date cost, which is the one with the smallest start time for job J_e. For any given J_e, J_t, and Y, we let $P(J_e, J_t, Y)$ denote the problem of determining a feasible solution (if any) with the smallest start time for J_e.

We define $P'(J_e, J_t, Y)$ as the problem of determining a feasible schedule (if any), such that J_e is the bottleneck early job, J_t is the bottleneck tardy job, the start time of J_e is $n-2$, the start time of J_t is $n+Y-2$, and that the start time of the first job in the schedule is maximized. Note that an optimal solution to problem $P'(J_e, J_t, Y)$ can be easily transformed into an optimal solution to problem $P(J_e, J_t, Y)$ by eliminating the idle time in front of the first job in the schedule (i.e., shifting all the jobs and the due-date to the left). Hence, we will focus on the development of an algorithm for problem $P'(J_e, J_t, Y)$.

To obtain an optimal solution to $P'(J_e, J_t, Y)$, we first use equation (1) to obtain the value of d. That is, $\alpha_e(d-n+1) = \beta_t(n+Y-1-d)$, which implies that $d = [(n-1)\alpha_e + (n+Y-1)\beta_t]/(\alpha_e+\beta_t)$. Next, we create $2n+Y-3$ positions on the machine, where the i^{th} position occupies the time slot $[i-1, i]$. Jobs J_e and J_t occupy the $(n-1)^{\text{st}}$ and $(n+Y-1)^{\text{st}}$ positions, respectively (see Figure 1). Let $R = \alpha_e(d-n+1)$, which is equal to the earliness cost of J_e as well as the tardiness cost of J_t. We would like to assign the jobs in $\{J_1, J_2, \ldots, J_n\} \setminus \{J_e, J_t\}$ to the remaining positions, such that the start time of the first job in the schedule is maximized, where J_j is permitted to occupy the k^{th} position if and only if $\max\{\alpha_j(d-k), \beta_j(k-d)\} \leq R$. This can be formulated as an $(n-2)\times(2n+Y-5)$ bipartite weighted matching problem and can be solved by some classical matching algorithms (see [2]). However, we propose a more efficient algorithm for this job assignment problem, which makes use of the special cost structure of the problem.

Lemma 2 *There exists an optimal solution to $P'(J_e, J_t, Y)$ in which all early or on-time jobs are sequenced in nondecreasing order of α_j.*

The proof of Lemma 2 follows a straightforward job interchange argument and is omitted. In what follows, we only consider solutions that satisfy the property stated in Lemma 2 (unless otherwise mentioned). Now, consider a subproblem of $P'(J_e, J_t, Y)$ with only jobs $\{J_1, J_2, \ldots, J_k\} \cup \{J_e, J_t\}$ and the first ℓ positions, where $k = 1, 2, \ldots, n$ and $\ell \in \{\lfloor d \rfloor+1, \lfloor d \rfloor+2, \ldots, 2n+Y-3\} \setminus$

$\{n+Y-1\}$, and $\lfloor d \rfloor$ denotes the largest integer less than or equal to d. We denote this subproblem as $P'_{k,\ell}(J_e, J_t, Y)$. If $\min_{j\in\{1,\ldots,k\}\setminus\{e,t\}}\{\beta_j\}\cdot(\ell-d) \le R$, then at least one job is permitted to occupy the last position (i.e., the ℓ^{th} position) of the schedule. In such a case, we define

$$j_0 = \arg \max_{j\in\{1,\ldots,k\}\setminus\{e,t\}} \{\alpha_j \mid \beta_j(\ell-d) \le R\},$$

with ties broken arbitrarily. That is, J_{j_0} is the job with the largest α value which is permitted to occupy the ℓ^{th} position.

Lemma 3 *If $P'_{k,\ell}(J_e, J_t, Y)$ is feasible, then there exists an optimal solution to $P'_{k,\ell}(J_e, J_t, Y)$ in which: (a) J_{j_0} is assigned to the ℓ^{th} position if $\min_{j\in\{1,\ldots,k\}\setminus\{e,t\}}\{\beta_j\}\cdot(\ell-d) \le R$; and (b) no job is assigned to the ℓ^{th} position otherwise.*

Proof Clearly, if $\min_{j\in\{1,\ldots,k\}\setminus\{e,t\}}\{\beta_j\}\cdot(\ell-d) > R$, then no job can be assigned to the ℓ^{th} position. Now, consider the case in which $\min_{j\in\{1,\ldots,k\}\setminus\{e,t\}}\{\beta_j\}\cdot(\ell-d) \le R$. Suppose that in an optimal solution, job J_{j_0} is assigned to the h^{th} position, where $h < \ell$. Let J_{j_1} be the job assigned to the ℓ^{th} position. We consider a new solution obtained by interchanging J_{j_0} and J_{j_1}. We show that this new solution remains feasible.

Case 1: $h > d$. In this case, both J_{j_0} and J_{j_1} are tardy jobs in the original solution. They remain tardy jobs in the new solution. Clearly, the tardiness cost of J_{j_1} has decreased. By definition of j_0, the tardiness cost of J_{j_0} in the new solution is no greater than R. Hence, the new solution is feasible.

Case 2: $h \le d$. In this case, J_{j_0} is an early (or on-time) job in the original solution. Thus, J_{j_1} becomes an early (or on-time) job in the new solution. By definition of j_0, we have $\alpha_{j_1} \le \alpha_{j_0}$. Hence, the earliness cost of J_{j_1} in the new solution is no greater than the earliness cost of J_{j_0} in the original solution. In addition, the tardiness cost of J_{j_0} in the new solution is no greater than R. Hence, the new solution is feasible.

Because the interchange of J_{j_0} and J_{j_1} does not affect the start time of the first job in the schedule, the new solution remains optimal. Note that this new solution does not necessarily satisfy the property stated in Lemma 2. However, we can further modify the new solution by resequencing the early jobs in nondecreasing order of α_j and having them occupy those positions that were originally occupied by early jobs. The modified solution must be feasible. ∎

Lemmas 2 and 3 suggest the following algorithm for problem $P'(J_e, J_t, Y)$. For $\ell = 2n+Y-3, 2n+Y-2, \ldots, \lfloor d \rfloor + 1$ (except for $\ell = n+Y-1$), check whether the unassigned job with the highest α value is permitted to occupy position ℓ, i.e., whether $\beta_j(\ell - d) \leq R$. If it is permitted, then assign that job to position ℓ; otherwise leave position ℓ unassigned. Next, we assign the remaining jobs to the early (or on-time) positions according to the property stated in Lemma 2 and make those early (or on-time) jobs start their processing as late as possible. If any of these early jobs has an earliness cost greater than R, then problem $P'(J_e, J_t, Y)$ is infeasible.

A straightforward implementation of the above algorithm requires $O(n^2)$ time. However, we provide a more efficient implementation, which enables us to solve problem $P'(J_e, J_t, Y)$ in $O(n \log n)$ time. As described above, we first assign jobs to the tardy positions. To simplify the notation, we denote the available tardy positions by

$$T \equiv \left\{\lfloor d \rfloor + 1, \lfloor d \rfloor + 2, \ldots, 2n + Y - 3\right\} \setminus \{n + Y - 1\},$$

and denote $|T|$ as the cardinality of set T. For a given job J_j, where $j \in \{1, 2, \ldots, n\} \setminus \{e, t\}$, we identify the maximal tardy position (if one exists), say position ℓ_j, such that J_j can be assigned to, that is, $\ell_j = \max\{k \in T \mid \beta_j(k - d) \leq R\}$. Therefore, the position for J_j is

$$\ell_j = \begin{cases} \left\lfloor d + \frac{R}{\beta_j} \right\rfloor, & \text{if } \left\lfloor d + \frac{R}{\beta_j} \right\rfloor \in T; \\ 2n + Y - 3, & \text{if } \left\lfloor d + \frac{R}{\beta_j} \right\rfloor > 2n + Y - 3; \\ n + Y - 2, & \text{if } \left\lfloor d + \frac{R}{\beta_j} \right\rfloor = n + Y - 1 \text{ and } n + Y - 2 \geq \lfloor d \rfloor + 1; \\ \text{undefined}, & \text{otherwise.} \end{cases} \tag{3}$$

(Note: If ℓ_j is undefined, then J_j cannot be assigned to any tardy position.) We keep the list of positions in a (job-indexed) array. In the next step, for each position $\ell \in T$, we identify all the jobs with ℓ being their maximal tardy position. This results in $|T|$ "reversed" (position-indexed) arrays, some of which may be empty. This step can be done simultaneously with the previous step using doubly-directed pointers. From each "reversed" array of position ℓ, we build a priority queue, denoted Q_ℓ, which holds the job indices in nonincreasing α value. Recall that a priority queue with n elements is a data structure that can be constructed in $O(n \log n)$ time, and it enables extraction of the maximal element (the *deQueue* operation) in $O(\log n)$ time (see, for example, [1]). Next, we assign jobs to the tardy positions in T as follows: Iterate ℓ from $2n+Y-3$ down to $\lfloor d \rfloor + 1$ (except for $\ell = n + Y - 1$). If Q_ℓ is empty, then no job can be assigned to position ℓ. In such a case, decrement ℓ (i.e., move to the next candidate tardy position). Otherwise, let $j_0 = \mathit{deQueue}(Q_\ell)$ (i.e., extract the job with the maximum α value that can be assigned to position ℓ), assign J_{j_0} to position ℓ, and remove it from the set of unassigned jobs. For the next iteration, add the elements of Q_ℓ to $Q_{\ell-1}$. We do this because Q_ℓ may contain a job with a larger α value than the jobs in $Q_{\ell-1}$, and the jobs in Q_ℓ can be assigned to any of the tardy positions $\ell, \ell-1, \dots, \lfloor d \rfloor+1$. Decrement ℓ and repeat the above procedure. This procedure is repeated until all the tardy positions have been considered.

The next phase is to assign the remaining jobs to the early or on-time positions $\{1, 2, \dots, \lfloor d \rfloor\} \setminus \{n-1\}$. This is done by assigning them sequentially to positions $\lfloor d \rfloor, \lfloor d \rfloor-1, \dots, n, n-2, \dots, 2, 1$ in nonincreasing α value. If any of these early jobs has an earliness cost greater than R, then problem $P'(J_e, J_t, Y)$ is infeasible. Algorithm Φ in Figure 2 is a straightforward implementation of the above idea.

We now argue that algorithm Φ can be implemented in $O(n \log n)$ time. Let n_ℓ denote the number of elements in Q_ℓ when Q_ℓ is first created, and let $n_{\lfloor d \rfloor} = 0$. In Figure 2, steps I4 and I5 take $O(n)$ time. For step I6, it takes $O(n)$ time to build the reversed arrays, and it takes $O(n_\ell \log n_\ell)$ time to build each priority queue Q_ℓ. Thus, the total running time of step I6 is

$$O(n) + \sum_{\ell \in T} O(n_\ell \log n_\ell) \le O(n) + O\Big(\sum_{\ell \in T} n_\ell \cdot \log n\Big) = O(n \log n).$$

Next, we consider the complexity of Phase A. The loop consists of $O(n)$ iterations. Step A3 takes $O(\log n)$ time, while steps A4 and A5 take constant time. As for step A6, we perform the union operation by inserting each element of $Q_{\ell-1}$ into Q_ℓ one by one. Each insertion operation takes $O(\log n)$ time. There are at most $n_{\ell-1}$ insertion operations for each $\ell \in T$. Thus, the contribution of step A6 to the total algorithm running time is

$$O\Big(\sum_{\ell \in T} n_{\ell-1} \cdot \log n\Big) \le O(n \log n).$$

Clearly, Phase B takes $O(n \log n)$ time. Hence, the total running time of algorithm Φ is $O(n \log n)$.

Note that the number of combinations of J_e, J_t, and Y is $O(n^3)$. Thus, under condition (iii) of Lemma 1, GMWAL(UJ) can be solved in $O(n^3) \cdot O(n \log n) = O(n^4 \log n)$ time.

Example. Consider an example with $n = 9$, $Y = 4$, $\alpha_e = \beta_t = 6$, and $\gamma = 0$. The other jobs, denoted $J_1, J_2, \dots, J_7$, have the following earliness and tardiness costs:

j	1	2	3	4	5	6	7
α_j	7	10	2	2	12	15	3
β_j	3	10	2	3	5	6	8

In this example, J_e and J_t are scheduled to positions 8 and 12, respectively. Note that $d = 10$, $R = 12$, and $2n + Y - 3 = 19$. Hence, the tardy positions are $\{11, 12, \dots, 19\}$, where position 12 has been assigned to J_t. Set S equals $\{1,2,3,4,5,6,7\}$ initially (see step I3 of algorithm Φ). The maximal tardy positions of jobs are given as:

$$\text{MaxTardyPosition}[1,2,\ldots,7] = (\ell_1,\ell_2,\ell_3,\ell_4,\ell_5,\ell_6,\ell_7) = (14,11,16,14,11,11,11).$$

Hence, we construct the following priority queues: $Q_{19}=\emptyset$, $Q_{18}=\emptyset$, $Q_{17}=\emptyset$, $Q_{16}=(3)$, $Q_{15}=\emptyset$, $Q_{14}=(1,4)$, $Q_{13}=\emptyset$, and $Q_{11}=(6,5,2,7)$.

We proceed to Phase A. Because $Q_{19}=Q_{18}=Q_{17}=\emptyset$, positions 19, 18, and 17 are left unoccupied. Next, we extract $j_0 = \mathit{deQueue}(Q_{16}) = 3$ and assign J_3 to position 16. After that, $Q_{16}=\emptyset$. Because $Q_{15}=\emptyset$, position 15 is left unoccupied. Next, we extract $j_0=\mathit{deQueue}(Q_{14})=1$ and assign J_1 to position 14. After that, $Q_{14}=(4)$, and we update $Q_{13} := Q_{13}\cup Q_{14} = (4)$. Next, we extract $j_0=\mathit{deQueue}(Q_{13})=4$ and assign J_4 to position 13. After that, $Q_{13}=\emptyset$. Next, we extract $j_0=\mathit{deQueue}(Q_{11})=6$ and assign J_6 to position 11. At the end of Phase A, the index set of unassigned jobs is $S=\{2,5,7\}$.

We now proceed to Phase B. We assign J_5, J_2, and J_7 to positions 10, 9, and 7, respectively. The resulting schedule is depicted in Figure 3(a), which is a feasible job assignment with the start time of the first job maximized. This is an optimal schedule for problem $P'(J_e,J_t,Y)$. The corresponding optimal schedule for problem $P(J_e,J_t,Y)$ is depicted in Figure 3(b). Note that an alternative optimal schedule can be obtained by changing the start time of J_3 from 9 to 8 (i.e., by eliminating the idle time between J_1 and J_3). ■

Finally, we consider condition (iv) of Lemma 1. If the optimal schedule satisfies condition (iv), then there exists a pair of "bottleneck tardy jobs" J_{t_1} and J_{t_2} with the same tardiness cost. Now, we present a solution method for the GMWAL(UJ) problem under condition (iv). We select (a) bottleneck tardy jobs J_{t_1} and J_{t_2} with $\beta_{t_1} > \beta_{t_2}$; and (b) the difference between the completion times of J_{t_1} and J_{t_2}, denoted as Y. Note that J_{t_2} must be processed after J_{t_1}. We determine if this combination of J_{t_1}, J_{t_2}, and Y yields a "feasible" solution, that is, a solution with J_{t_1} and J_{t_2} being the bottleneck tardy jobs. If it yields a feasible solution, then we determine the smallest start time for J_{t_1} such that a feasible solution exists. We repeat this procedure with all possible combinations of J_{t_1}, J_{t_2}, and Y. (Again, it suffices to consider $Y=1,2,\ldots,n-1$.) Among all feasible solutions generated, we select the one with the lowest total cost. For any given J_{t_1}, J_{t_2}, and Y, we let $\bar{P}(J_{t_1},J_{t_2},Y)$ denote the problem of determining a feasible solution (if any) with the smallest start time for J_{t_1}.

We define $\bar{P}'(J_{t_1}, J_{t_2}, Y)$ as the problem of determining a feasible schedule (if any), such that J_{t_1} and J_{t_2} are the bottleneck tardy jobs, the start time of J_{t_1} is $n-2$, the start time of J_{t_2} is $n+Y-2$, and that the start time of the first job in the schedule is maximized. As in the case of problem $P(J_e, J_t, Y)$, an optimal solution to problem $\bar{P}'(J_{t_1}, J_{t_2}, Y)$ can be easily transformed into an optimal solution to problem $\bar{P}(J_{t_1}, J_{t_2}, Y)$ by eliminating the idle time in front of the first job in the schedule. Hence, we will focus on the development of an algorithm for problem $\bar{P}'(J_{t_1}, J_{t_2}, Y)$.

To obtain an optimal solution to $\bar{P}'(J_{t_1}, J_{t_2}, Y)$, we first use equation (2) to obtain the value of d. That is, $\beta_{t_1}(n-1-d) = \beta_{t_2}(n+Y-1-d)$, which implies that $d = [(n-1)\beta_{t_1} - (n+Y-1)\beta_{t_2}]/(\beta_{t_1} - \beta_{t_2})$. If $d < 0$, then $\bar{P}'(J_{t_1}, J_{t_2}, Y)$ is infeasible, which implies that the current combination of J_{t_1}, J_{t_2}, and Y cannot yield an optimal solution to GMWAL(UJ). If $d \geq 0$, then we create $2n+Y-3$ positions on the machine as shown in Figure 4. Let $R = \beta_{t_1}(n-1-d)$. We would like to assign the jobs in $\{J_1, J_2, \ldots, J_n\} \setminus \{J_{t_1}, J_{t_2}\}$ to the remaining positions, where J_j is permitted to occupy the k^{th} position if and only if $\max\{\alpha_j(d-k), \beta_j(k-d)\} \leq R$. It is easy to check that Lemma 2 is also valid for problem $\bar{P}'(J_{t_1}, J_{t_2}, Y)$.

Define $\bar{P}'_{k,\ell}(J_{t_1}, J_{t_2}, Y)$ as a subproblem of $\bar{P}'(J_{t_1}, J_{t_2}, Y)$ with only jobs $\{J_1, J_2, \ldots, J_k\} \cup \{J_{t_1}, J_{t_2}\}$ and the first ℓ positions. It is easy to check that Lemma 3 is also valid for problem $\bar{P}'_{k,\ell}(J_{t_1}, J_{t_2}, Y)$. Thus, starting with $S = \{1, 2, \ldots, n\} \setminus \{t_1, t_2\}$, we can use Phases A and B of algorithm Φ (see Figure 2) to solve problem $\bar{P}'(J_{t_1}, J_{t_2}, Y)$. Thus, under condition (iv) of Lemma 1, GMWAL(UJ) can be solved in $O(n^4 \log n)$ time.

3 Conclusion

We have shown that the problem of minimizing maximum weighted earliness-tardiness and due-date costs on a single machine can be solved in $O(n^4 \log n)$ time when all jobs have identical processing times. Our algorithm is based on the enumeration of a pair of bottleneck jobs (either a bottleneck early job and a bottleneck tardy job, or a pair of bottleneck tardy jobs) as well as the difference of their completion times. Based on the optimal properties presented in Lemmas 2 and 3, as well as the use of priority queues, we have developed an efficient method to assign the non-bottleneck jobs. It remains a challenging question of whether a complete enumeration of the bottleneck jobs and the difference of their start times can be avoided so that a further reduction in complexity of the algorithm can be achieved.

Acknowledgements

The authors would like to thank two anonymous referees for their helpful comments and suggestions. This research was supported in part by the Recanati Fund of The School of Business Administration, The Hebrew University, Jerusalem, Israel and by Grant PolyU 5222/06E from the Hong Kong Research Grants Council.

References

[1] Aho, A.V., J.E. Hopcroft and J.D. Ullman, 1974. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA.

[2] Ahuja, R.K., T.L. Magnanti and J.B. Orlin, 1993. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Englewood Cliffs, NJ.

[3] Gordon, V., J.M. Proth and C. Chu, 2002. A survey of the state-of-the-art of common due date assignment and scheduling research. *European Journal of Operational Research* **139**, 1-25.

[4] Li, C.-L. and T.C.E. Cheng, 1994. The parallel machine min-max weighted absolute lateness scheduling problem. *Naval Research Logistics* **41**, 33-46.

[5] Mosheiov, G. and U. Yovel, 2006. Minimizing weighted earliness-tardiness and due-date costs with unit processing-time jobs. *European Journal of Operational Research,* **172**, 528-544.

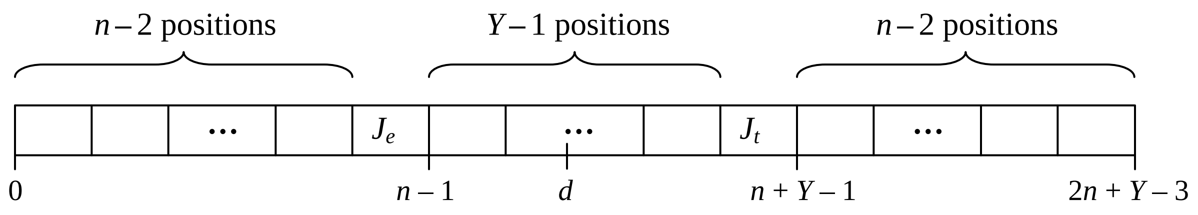
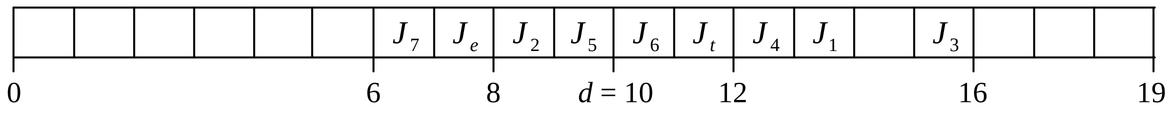


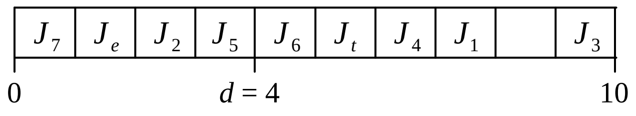
Figure 1. The positions for job assignment in problem $P'(J_e, J_t, Y)$

```
ALGORITHM $\Phi$
Input: $n$, $\alpha_1, \ldots, \alpha_n$, $\beta_1, \ldots, \beta_n$, $J_e$, $J_t$, $Y$.
Output: A feasible job assignment (if one exists) with $J_e$ in position $n-1$, $J_t$ in position $n+Y-1$,
        and the start time of the first job maximized.

BEGIN
INITIALIZATION PHASE
I1. $d := [(n-1)\alpha_e + (n+Y-1)\beta_t]/(\alpha_e + \beta_t)$;
I2. $R := \alpha_e(d - n + 1)$;
I3. $S := \{1, 2, \ldots, n\} \setminus \{e, t\}$ = set of indices of unassigned jobs.
I4. for $j = 1$ to $n$   (except for $j = e, t$):
I5.       MaxTardyPosition[$j$] := $\ell_j$, where $\ell_j$ is defined in equation (3);
I6. build reversed arrays and priority queues:
          for each $\ell \in T$, build an array of job indices $j$ such that MaxTardyPosition[$j$] = $\ell$; from
          each such a (possibly empty) array, build a priority queue $Q_\ell$ of the indices sorted by
          nonincreasing $\alpha$ value.

PHASE A: (Assign jobs to tardy positions)
A1. for $\ell = 2n + Y - 3$ down to $\lfloor d \rfloor + 1$:
A2.       if position $\ell$ is unoccupied and $Q_\ell \neq \emptyset$ then
A3.            $j_0 := deQueue(Q_\ell)$;
A4.            assign $J_{j_0}$ to position $\ell$;
A5.            $S := S \setminus \{j_0\}$;
A6.            $Q_{\ell-1} := Q_{\ell-1} \cup Q_\ell$ (performed by inserting each element of $Q_{\ell-1}$ into $Q_\ell$ one by one).

PHASE B: (Assign remaining jobs to early or on-time positions)
B1. sort and re-index the remaining jobs as $J_1, J_2, \ldots, J_u$, where $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_u$;
B2. $pos := \lfloor d \rfloor$;
B3. for $j = 1$ to $u$:
B4.       if position $pos$ is unoccupied then
B5.            if $\alpha_j(d - pos) \leq R$ then assign $J_j$ to position $pos$;
B6.            otherwise stop; the problem is infeasible;
B7.       $pos := pos - 1$.
END
```

Figure 2. Algorithm Φ for problem $P'(J_e, J_t, Y)$



(a) Optimal schedule for problem $P'(J_e, J_t, Y)$



(b) Optimal schedule for problem $P(J_e, J_t, Y)$

Figure 3. Schedule for the example

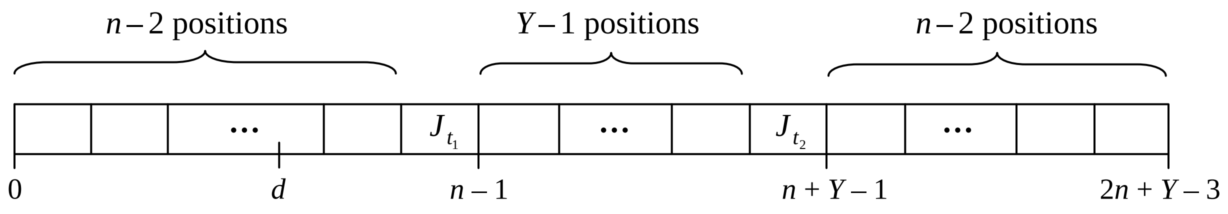


Figure 4. The positions for job assignment in problem $\overline{P}'(J_{t_1}, J_{t_2}, Y)$