WORKING PAPER

AN EFFICIENT ALGORITHM FOR A BICRITERIA SINGLE-MACHINE SCHEDULING PROBLEM

Naoki Katoh

October 1987
WP-87-100

International Institute
for Applied Systems Analysis
A-2361 Laxenburg/Austria

IIASA

An Efficient Algorithm for a Bicriteria Single-Machine Scheduling Problem

Naoki Katoh

October 1987
WP-87-100

INTERNATIONAL INSTITUTE FOR APPLIED SYSTEMS ANALYSIS
A-2361 Laxenburg, Austria

Foreword

This paper considers a single-machine scheduling problem in which the maximum tardiness and the total flowtime are two objectives to be minimized. Instead of enumerating all efficient schedules, the author considers the problem of minimizing the maximum of the weighted values of these two objective functions, which arises in interactive multicriteria decision making. The author proposes a strongly polynomial algorithm for this problem which runs in $O(n^2 \log n)$ time.

Alexander B. Kurzhanski
Chairman
System and Decision Sciences program

An Efficient Algorithm for a Bicriteria Single-Machine Scheduling Problem

Naoki Katoh

1. Introduction

It was reported in Panwalkar et al. [10] that in industrial scheduling managers do schedule according to multiple objectives. A schedule which is optimum with respect to one criterion normally performs badly with respect to other criteria. Therefore a schedule with satisfactory performance on all measures may be considered as a better alternative for the decision maker. This consideration leads to the research on multicriteria scheduling which has recently been done by several researchers (see Chapter 6 of the book by Blazewicz et al. [2] for the survey of this topic).

This paper is concerned with a single-machine scheduling problem with two criteria, i.e., the maximum tardiness and the total flowtime. This problem is called a *bicriteria single-machine scheduling problem* and is formulated as follows. We are given n jobs that are to be sequenced on a single machine. Jobs are numbered from 1 through n. Each job j has processing time p_j and due date d_j. p_j and d_j are assumed to be nonnegative integers. All jobs are assumed to be available at time 0. Let

Π: the set of permutation schedules,

π: a permutation schedule,

$C_j(\pi)$: the completion time of job j for a given schedule π.

$T_j(\pi) \equiv \max\{0, C_j(\pi) - d_j\}$: the tardiness of job j for a given schedule π.

$T_{\max}(\pi) \equiv \max_{1 \le j \le n} T_j(\pi)$: the maximum tardiness of a given schedule π.

$F(\pi) \equiv \sum_{j=1}^{n} C_j(\pi)$: the total flowtime of a given schedule π.

Then the problem is formulated as

$$\underset{\pi \in \Pi}{\text{minimize}} \{T_{\max}(\pi),\ F(\pi)\} \quad . \tag{1}$$

Since both objectives cannot be minimized simultaneously in general, it is quite natural that the decision maker chooses an efficient schedule as his decision. Here a schedule $\pi^* \in \Pi$ is said to be *efficient* for the problem (1) if there exists no $\pi \in \Pi$ such that

$$T_{\max}(\pi) \leq T_{\max}(\pi^*) \quad \text{and} \quad F(\pi) \leq F(\pi^*)$$

hold and at least one relation holds with strict inequality. Hence it is important to determine all efficient schedules. This problem has been studied by Wassenhove and Gelders [15] and Nelson, Sarin and Daniels [9]. [15] proposed a pseudo-polynomial algorithm with $O(n \log n \cdot \sum_{j=1}^{n} p_j)$ running time, and presented some computational results. [9] also did the similar work. In addition, [9] considered the other types of bicriteria single machine scheduling problems.

Though it was shown by the computational experiments done in [9, 15] that finding all efficient schedules for problem (1) can be done quite efficiently for problem instances with the number of jobs being up to 50, their approach has two drawbacks. One is that the running time required for the algorithm is not polynomially bounded and the second is that it may generate so many efficient schedules and thus this may be confusing for the decision maker because he or she must choose one best schedule according to his or her preference from among a large number of efficient schedules. With this consideration, we shall take the following alternative approach, which has been used in interactive multicriteria decision making. It solves the following problem P instead of enumerating all efficient schedules.

$$P: \quad \underset{\pi \in \Pi}{\text{minimize}}\ z(\pi) \equiv \max\{\alpha_1 T_{\max}(\pi) + \beta_1,\ \alpha_2 F(\pi) + \beta_2\} \quad , \tag{2}$$

where α_i and β_i are positive and real constants respectively, which are directly specified by the decision maker or are determined based on the information supplied by the decision maker.

α_i and β_i are typically determined in the following manner by the quasisatisficing method, which is one of the well known methods used in interactive multicriteria decision making (see Wierzbicki and Lewandowski [21] and Chapter 7 of the book by Sawaragi, Nakayama and Tanino [11]). The method first requires the decision maker to specify the aspiration level q_i and the reservation level r_i for $i = 1,2$. q_1 (resp. q_2) is interpreted as

the desirable value for the maximum tardiness (resp. the total flowtime) that the decision maker would like to attain, and r_1 (resp. r_2) is the maximum allowable value for the maximum tardiness (resp. the total flowtime). The degree of the achievement of a given $\pi \in \Pi$ for the objective of the maximum tardiness (resp. the total flowtime) is measured by

$$\mu_1(q_1,r_1, T_{\max}(\pi)) = (r_1 - T_{\max}(\pi))/(r_1 - q_1) \quad , \tag{3}$$

$$(\text{resp. } \mu_2(q_2,r_2,F(\pi)) = (r_2 - F(\pi))/(r_2 - q_2)) \quad . \tag{4}$$

The aggregated degree of the achievement for π is then measured by

$$s \equiv \min \{\mu_1(q_1,r_1,T_{\max}(\pi)), \mu_2(q_2,r_2,F(\pi))\} \quad . \tag{5}$$

The method then solves the following problem:

$$\underset{\pi\in\Pi}{\text{maximize}}\ s \quad , \tag{6}$$

and provides its optimal solution π^* to the decision maker. If π^* is not satisfactory for the decision maker, he or she is asked to change the aspiration and/or reservation levels in order to obtain a satisfactory alternative. The above process is repeated until a satisfactory schedule is obtained. At each round of this iteration, we need to solve the problem (6). Letting $\alpha_i = 1/(r_i - q_i)$ and $\beta_i = -r_i/(r_i - q_i)$, the problem (6) is equivalent to Problem P.

Modifications and generalizations of the achievement function s in (5) have been proposed by several authors, e.g., Wierzbicki [16, 17, 18, 19, 20], Nakayama [8], Steuer and Choo [13] (see also [21] for general discussion about this subject). Many of those have the form similar to the one in (5).

In view of this, it is important to study the computational complexity for solving Problem P. The aim of this paper is to present a strongly polynomial algorithm for Problem P. An algorithm that solves a problem whose input consists of m real numbers is ***strongly polynomial*** (see [7, 14]) if

(a) it performs only elementary arithmetic operations (additions, subtractions, comparisons, multiplications and divisions),

(b) the number of operations required to solve the problem is polynomially bounded in m, and

(c) when applied to rational data, the size of the numbers (i.e., the number of bits required to represent the numbers) that the algorithm generates is polynomially bounded in m and the size of the input numbers.

It is shown that our algorithm requires $O(n^2 \log n)$ time. This time bound is achieved by applying Megiddo's ingeneous work [6] which was originally developed for solving combinatorial fractional programs. Gusfield [3] applied Megiddo's idea to the other type of combinatorial optimization problems. However, the application of Mediddo's idea to our problem seems to be new.

This paper is organized as follows. Section 2 gives basic properties on which our algorithm is based. Section 3 describes the algorithm, proves its correctness and analyzes its running time.

2. Basic Properties

It is clear that a schedule π^* optimal to Problem P exists among all efficient schedules (see [11, 20, 21]). It is shown in [15] that the set of all efficient schedules is determined by solving the following parametric scheduling problem $P(\lambda)$ for all λ with $t_{\min} \leq \lambda \leq \sum_{j=1}^{n} p_j$, where $t_{\min}$ is defined by

$$t_{\min} = \min_{\pi \in \Pi} T_{\max}(\pi) \quad .$$

$$P(\lambda): \ \underset{\pi \in \Pi}{\text{minimize}} \ F(\pi) = \sum_{j=1}^{n} C_j(\pi) \tag{7}$$

$$\text{subject to } C_j(\pi) \leq d_j + \lambda, \ j \in \{1,2,...,n\} \quad . \tag{8}$$

This problem is known as the single-machine scheduling problem for minimizing the total flowtime with deadlines, and Smith [12] proposed an $O(n \log n)$ time algorithm for solving this problem. His algorithm is called a Smith's *backward scheduling rule*. It first determines the job located in the last position in an optimal schedule π^*, secondly the one in the second last position in π^*, and so on. We shall describe his algorithm because it will be used later as a subroutine in the algorithm for Problem P proposed in Section 3.

Procedure SOLVEDEAD(λ)

Input: The set of jobs $\{1,2,...,n\}$ with processing times p_j and due dates d_j, and parameter λ with $t_{\min} \leq \lambda \leq \sum_{j=1}^{n} p_j$.

Output: An optimal schedule for Problem $P(\lambda)$.

Step 1: Let $R := \sum_{j=1}^{n} p_j, S := \{1,...,n\}$, $k := n$.

Step 2: Find j^* such that

$$p_{j^*} = \max \{p_j | j \in S, d_j + \lambda \geq R\} \quad .$$

Ties are broken by choosing the one with the largest due date. Assign job j^* to position k.

Step 3: Let $R := R - p_{j^*}$, $S := S - \{j^*\}$, $k := k - 1$. If $k = 0$, halt (the optimal schedule is obtained). Else return to Step 2. □

Lemma 1 [15] Procedure SOLVEDEAD(λ) correctly solves Problem $P(\lambda)$ in $0(n \log n)$ time and the obtained schedule is efficient. □

Let $\pi(\lambda)$ denote an optimal schedule for $P(\lambda)$ which is obtained by Procedure SOLVEDEAD (λ).

Lemma 2. $F(\pi(\lambda))$ is nonincreasing in λ and $T_{\max}(\pi(\lambda))$ is nondecreasing in λ.

Proof. If $F(\pi(\lambda))$ is not nonincreasing in λ, there exist λ and λ' with $\lambda < \lambda'$ such that $F(\pi(\lambda)) < F((\pi(\lambda'))$. By (8), any schedule feasible to $P(\lambda)$ is also feasible to $P(\lambda')$. This contradicts the optimality of $\pi(\lambda')$ by $F(\pi(\lambda)) < F(\pi(\lambda'))$. If $T_{\max}(\pi(\lambda))$ is not nondecreasing in λ, there exist λ and λ' with $T_{\max}(\pi(\lambda)) > T_{\max}(\pi(\lambda'))$. Since $F(\pi(\lambda)) \geq F(\pi(\lambda'))$ holds as proved above, it follows that $\pi(\lambda)$ is not efficient. This contradicts Lemma 1. □

The following lemma is important for constructing our algorithm. Define

$$p_{sum} = \sum_{j=1}^{n} p_j \quad . \tag{9}$$

Lemma 3. (i) If $\pi(t_{\min})$ satisfies

$$\alpha_1 T_{\max}(\pi(t_{\min})) + \beta_1 \geq \alpha_2 F(\pi(t_{\min})) + \beta_2 \quad , \tag{10}$$

$\pi(t_{\min})$ is optimal to P.

(ii) If $\pi(p_{sum})$ satisfies

$$\alpha_1 T_{\max}(\pi(p_{sum})) + \beta_1 \leq \alpha_2 F(\pi(p_{sum})) + \beta_2 \quad , \tag{11}$$

$\pi(p_{sum})$ is optimal to P.

(iii) If neither the condition of (i) nor (ii) holds, let

$$\lambda_{\min} = \min\{\lambda | t_{\min} \leq \lambda \leq p_{sum},\ \alpha_1 T_{\max}(\pi(\lambda)) + \beta_1 \geq \alpha_2 F(\pi(\lambda)) + \beta_2\} \quad , \tag{12}$$

and let λ^* be one of $\lambda_{\min}$ and $\lambda_{\min}-1$ such that

$$z(\pi(\lambda^*)) = \min\{z(\pi(\lambda_{\min})),\ z(\pi(\lambda_{\min}-1))\} \quad , \tag{13}$$

where $z(\cdot)$ is defined in (2). Then $\pi(\lambda^*)$ is optimal to P.

Proof. (i) If (10) holds, we have from Lemma 2 that

$$\alpha_1 T_{\max}(\pi(\lambda)) + \beta_1 \geq \alpha_2 F(\pi(\lambda)) + \beta_2, \text{ for all } \lambda \text{ with } t_{\min} \leq \lambda \leq p_{sum}$$

and

$$T_{\max}(\pi(t_{\min})) = \min\{T_{\max}(\pi(\lambda)) | t_{\min} \leq \lambda \leq p_{sum}\}$$

hold. It implies that $z(\pi(t_{\min})) = \min\{z(\pi(\lambda)) | t_{\min} \leq \lambda \leq p_{sum}\}$ holds by definition of $z(\cdot)$.

(ii) If (11) holds, we have from Lemma 2 that

$$\alpha_1 T_{\max}(\pi(\lambda)) + \beta_1 \leq \alpha_2 F(\pi(\lambda)) + \beta_2, \text{ for all } \lambda \text{ with } t_{\min} \leq \lambda \leq p_{sum}$$

and

$$F(\pi(p_{sum})) = \min\{F(\pi(\lambda)) | t_{\min} \leq \lambda \leq p_{sum}\}$$

hold. It implies that $z(\pi(p_{sum})) = \min\{z(\pi(\lambda)) | t_{\min} \leq \lambda \leq p_{sum}\}$ holds by definition of $z(\cdot)$.

(iii) We shall first show that $\lambda_{\min}$ defined in (12) always exists if neither the condition of (i) nor (ii) holds. Let $\tilde{\pi} \in \Pi$ be defined by

$$\begin{aligned} T_{\max}(\tilde{\pi}) = \min\{T_{\max}(\pi) | \pi \text{ is efficient and } \alpha_1 T_{\max}(\pi) + \beta_1 \\ \geq \alpha_2 F(\pi) + \beta_2\} \quad . \end{aligned} \tag{14}$$

$\tilde{\pi}$ always exists since $\pi(p_{sum})$ satisfies the condition of the right-hand side of (14) by assumption and Π is a finite set. We shall show that $\lambda_{\min} = T_{\max}(\tilde{\pi})$ holds. Note that $\tilde{\pi}$ is optimal to $P(\tilde{\lambda})$ with $\tilde{\lambda} = T_{\max}(\tilde{\pi})$ since $\tilde{\pi}$ is efficient. Then, $\pi(\lambda)$ for any $\lambda < \tilde{\lambda}$ satisfies $\alpha_1 T_{\max}(\pi(\lambda)) + \beta_1 < \alpha_2\ F(\pi(\lambda)) + \beta_2$, since otherwise there exists λ' with $\lambda' < \tilde{\lambda}$ with $\alpha_1 T_{\max}(\pi(\lambda')) + \beta_1 \geq \alpha_2 F(\pi(\lambda')) + \beta_2$. This contradicts the definition of $\tilde{\pi}$ since $T_{\max}(\pi(\lambda')) \leq \lambda' (< \tilde{\lambda})$ holds by (8). Therefore $\lambda_{\min} = T_{\max}(\tilde{\pi})$ follows.

We shall then show that $z(\pi(\lambda_{\min})) \leq z(\pi(\lambda))$ holds for all λ with $\lambda \geq \lambda_{\min}$. Since $T_{\max}(\pi(\lambda))$ is nondecreasing in λ by Lemma 2 and $\alpha_1 T_{\max}(\pi(\lambda)) + \beta_1 \geq \alpha_2 F(\pi(\lambda)) + \beta_2$ holds for all λ with $\lambda \geq \lambda_{\min}$ by (12), $z(\pi(\lambda_{\min})) \leq z(\pi(\lambda))$ follows. Note that $T_{\max}(\pi)$ takes only integer values since all p_j and d_j are assumed to be integers. Together with Lemma 2, this implies

$$T_{\max}(\pi(\lambda_{\min} - 1)) = \max \{ T_{\max}(\pi(\lambda)) | t_{\min} \leq \lambda < \lambda_{\min} \} \quad \textit{and} \tag{15}$$

$$F(\pi(\lambda_{\min} - 1)) = \min \{ F(\pi(\lambda)) | t_{\min} \leq \lambda < \lambda_{\min} \} \quad . \tag{16}$$

Since $\alpha_1 T_{\max}(\pi(\lambda)) + \beta_1 < \alpha_2 F(\pi(\lambda)) + \beta_2$ holds for all λ with $t_{\min} \leq \lambda < \lambda_{\min}$ by (12), it follows from (16) that $z(\pi(\lambda_{\min} - 1)) \leq z(\pi(\lambda))$ holds for all λ with $\lambda < \lambda_{\min}$. Therefore Lemma 3 (iii) follows. □

3. A Strongly Polynomial Algorithm for P

We shall first explain the outline of the algorithm. It first solves $P(t_{\min})$ and $P(p_{sum})$ to test whether the condition of Lemma 3 (i) or Lemma 3 (ii) holds. If the condition of Lemma 3 (i) (resp. Lemma 3 (ii)) holds, a schedule $\pi(t_{\min})$ (resp. $\pi(p_{sum})$) is output as an optimal schedule to Problem P. We assume in the following discussion that neither the condition of Lemma 3 (i) nor Lemma 3 (ii) holds, i.e., the condition of Lemma 3 (iii) holds. In this case, we need to compute $\lambda_{\min}$ in (12). First notice that $\lambda_{\min}$ can be found by applying the binary search over the interval $[t_{\min}, p_{sum}]$. We first try the value of $\lambda = \lfloor \frac{t_{\min} + p_{sum}}{2} \rfloor$ to test whether $\alpha_1 T_{\max}(\pi(\lambda)) + \beta_1 \geq \alpha_2 F(\pi(\lambda)) + \beta_2$ holds or not. Here $\lfloor x \rfloor$ denotes the integer part of x. If it holds, $\lambda_{\min}$ is contained in the interval $[\lambda, p_{sum}]$. Otherwise it is contained in $[\lambda_{\min}, \lambda]$. In any case, we can halve the interval. After k trial values the length of the remaining interval can be no greater than $(p_{sum} - t_{\min})/2^k$. We continue the interval-halving procedure until the remaining interval has the length smaller than one , since $P(\lambda)$ and $P(\lambda')$ for λ and λ' with $\lambda \neq \lambda'$ and $\lfloor \lambda \rfloor = \lfloor \lambda' \rfloor$ have the same set of optimal schedules by the integrality of p_j and d_j. Therefore $\lambda_{\min}$ can be found in $O(n \log n \cdot \log p_{sum})$ time and as a result of Lemma 3 (iii) an optimal schedule of Problem P can be found in $O(n \log n \cdot \log p_{sum})$ time. This algorithm is, however, not strongly polynomial because of the term $\log p_{sum}$.

In order to achieve a strongly polynomial algorithm for P, we employ Megiddo's idea [6] which was originally developed for solving combinatorial fractional programs. The algorithm applies Procedure SOLVEDEAD($\lambda_{\min}$). The computation path of

SOLVEDEAD($\lambda_{\min}$) may contain conditional jump operations, each of which selects proper computation path depending upon the outcome of comparing two numbers. Notice that SOLVEDEAD($\lambda_{\min}$) contains arithmetic operations of only additions and subtractions, and comparisons of the numbers generated from the given problem data. Thus, when applying SOLVEDEAD(λ) to solve $P(\lambda)$ with λ treated as unknown parameter, the numbers generated in the algorithm are all linear functions of λ or constants. Note that comparisons are necessary at conditional jumps. If a comparison for a conditional jump operation is made between two linear functions of $\lambda_{\min}$, the condition can be written in the form of

$$\lambda_{\min} > \tilde{\lambda},\ \lambda_{\min} = \tilde{\lambda} \ \text{ or } \ \lambda_{\min} < \tilde{\lambda} \tag{17}$$

for an appropriate critical value $\tilde{\lambda}$, which can be determined by solving the linear equation in $\lambda_{\min}$ constructed from the compared two linear functions.

An important observation here is that the condition (17) can be tested without knowing the value of $\lambda_{\min}$. This is carried out as follows. The algorithm starts with the interval $(\underline{\lambda}, \bar{\lambda}]$, where $\underline{\lambda} = t_{\min}$ and $\bar{\lambda} = p_{sum}$. If $\hat{\lambda} < \underline{\lambda}$ (resp. $\hat{\lambda} > \bar{\lambda}$), it is concluded that $\hat{\lambda} < \lambda_{\min}$ (resp. $\hat{\lambda} > \lambda_{\min}$) holds. Otherwise (i.e., $\underline{\lambda} \le \hat{\lambda} \le \bar{\lambda}$), $P(\hat{\lambda})$ and $P(\hat{\lambda}-1)$ are solved by calling procedures SOLVEDEAD($\hat{\lambda}$) and SOLVEDEAD($\hat{\lambda}-1$) respectively. If

$$\alpha_1 T_{\max}(\pi(\hat{\lambda}-1)) + \beta_1 \ge \alpha_2 F(\pi(\hat{\lambda}-1)) + \beta_2 \tag{18}$$

holds, $\hat{\lambda} > \lambda_{\min}$ is concluded since Lemma 2 implies $\alpha_1 T_{\max}(\pi(\hat{\lambda})) + \beta_1 \ge \alpha_2 F(\pi(\hat{\lambda})) + \beta_2$, and the algorithm follows the corresponding proper computation path. If

$$\begin{aligned} &\alpha_1 T_{\max}(\pi(\hat{\lambda})) + \beta_1 \ge \alpha_2 F(\pi(\hat{\lambda})) + \beta_2 \\ &\textit{and}\ \ \alpha_1 T_{\max}(\pi(\hat{\lambda}-1)) + \beta_1 < \alpha_2 F(\pi(\hat{\lambda}-1)) + \beta_2 \ \ , \end{aligned} \tag{19}$$

$\hat{\lambda} = \lambda_{\min}$ is concluded since $\hat{\lambda}$ is an integer as discussed above and $\lambda_{\min}$ is also an integer as seen from the proof of Lemma 3. Then the algorithm outputs π^* which is chosen from $\pi(\lambda_{\min})$ and $\pi(\lambda_{\min}-1)$ by (13) and halts. Finally if

$$\alpha_1 T_{\max}(\pi(\hat{\lambda})) + \beta_1 < \alpha_2 F(\pi(\hat{\lambda})) + \beta_2 \tag{20}$$

holds, $\hat{\lambda} < \lambda_{\min}$ is concluded and the algorithm follows the corresponding computation path. We shall show later that $\lambda_{\min}$ is found among critical values generated during the course of the algorithm.

Since SOLVEDEAD(λ) requires $O(n\log n)$ number of jump operations as shown in [15], and SOLVEDEAD($\hat{\lambda}$) and SOLVEDEAD($\hat{\lambda}-1$) need be solved for a critical value $\hat{\lambda}$ at each jump operation, our algorithm requires $O(n^2\log^2 n)$ time in total.

This time bound is further improved to $O(n^2\log n)$ by showing that the above algorithm can be implemented so that the number of comparisons in which at least one of compared two numbers contains $\lambda_{\min}$ can be bounded by $O(n)$. Notice that the critical values are generated only when Step 2 tests whether $d_j+\lambda_{\min}\geq R$ holds or not for each job $j\in S$ (i.e., compared two numbers are constants in other comparisons), and that the critical value for job j is $R-d_j$. By the way of updating R and the integrality of p_j and $d_j, R-d_j$ takes only integers. If (20) holds for $\hat{\lambda}=R-d_j, j$ is added to the set over which the maximum is taken in Step 2, since $\hat{\lambda}=R-d_j<\lambda_{\min}$ holds. If (18) holds, j is not added to the set. If (19) holds, the algorithm outputs π^* as explained before and halts. Letting

$$J(S,R)=\{j\in S\,|\,\hat{\lambda}=R-d_j \text{ satisfies (20)}\} \tag{21}$$

$$p_{j^*}=\max\{p_j\,|\,j\in J(S,R)\}\quad , \tag{22}$$

the position in schedule $\pi(\lambda_{\min})$ of job j^* is determined, i.e., j^* is placed so that its completion time $C_{j^*}(\pi(\lambda_{\min}))$ is equal to R. After this, R and S are updated as $R'=R-p_{j^*}$ and $S'=S-\{j^*\}$. Then $J(S',R')$ and p_{j^*} are recomputed. An important observation here is that once a job j is added to $J(S,R)$, it is again added to $J(S',R')$ at the next round unless j is chosen as j^* in (22), since, if (20) is satisfied for $\hat{\lambda}=R-d_j$, it is again satisfied for the new critical value $\hat{\lambda}'=R'-d_j(<R-d_j)$ by Lemma 2. This reduces the number of times to test the condition of (20) to $O(n)$. The following is the description of the algorithm for solving Problem P. In the algorithm, J and s serve as the set $J,(S,R)$ of (21) and the suffix of $\pi(\lambda_{\min})$ currently obtained, respectively.

Procedure SOLVEP

Input: Job processing times $p_j, j=1,...,n$, due dates $d_j, j=1,...,n$, and the weights $\alpha_j(>0)$ and $\beta_j, j=1,...,n$.

Output: An optimal schedule of Problem P.

Step 0: Let $R:=p_{sum}:=\sum_{j=1}^{n}p_j$. Solve the problem of minimizing the maximum tardiness and let $t_{\min}$ be its optimal objective value.

Step 1: (i) Call Procedure SOLVEDEAD($t_{\min}$) to find $\pi(t_{\min})$. If $\pi(t_{\min})$ satisfies (10), output $\pi(t_{\min})$ is an optimal schedule to P and halt (Lemma 3 (i)).

(ii) Call Procedure SOLVEDEAD(p_{sum}) to find $\pi(p_{sum})$. If $\pi(p_{sum})$ satisfies (11), output $\pi(p_{sum})$ as an optimal schedule to P and halt (Lemma 3 (ii)).

Step 2: Let $\underline{\lambda} := t_{\min}$ and $\bar{\lambda} := p_{sum}$. Rearrange the indices of jobs so that $d_1 \geq d_2 \geq \cdots \geq d_n$ holds. Let $J := \emptyset$, $s := \Lambda$ and $k := 1$.

Step 3: If $k = n+1$, go to Step 5. Else go to Step 4.

Step 4: Let $\hat{\lambda} := R - d_k$.

(i) If $\hat{\lambda} < \underline{\lambda}$, let $J := J \cup \{k\}$ and $k := k+1$. Return to Step 3.

(ii) If $\hat{\lambda} > \bar{\lambda}$, go to Step 5.

(iii) If $\underline{\lambda} \leq \hat{\lambda} \leq \bar{\lambda}$, call Procedures SOLVEDEAD ($\hat{\lambda}$) and SOLVEDEAD ($\hat{\lambda} - 1$) to compute $\pi(\hat{\lambda})$ and $\pi(\hat{\lambda} - 1)$ respectively.

(iii-a) If (20) holds, let $\underline{\lambda} := \hat{\lambda}$, $J := J \cup \{k\}$ and $k := k+1$. Return to Step 3.

(iii-b) If (19) holds, Let π^* be one of $\pi(\hat{\lambda})$ and $\pi(\hat{\lambda} - 1)$ satisfying

$$z(\pi^*) = \min\{z(\pi(\hat{\lambda})), z(\pi(\hat{\lambda} - 1))\} \quad .$$

Output π^* as an optimal schedule to P (Lemma 3 (iii)) and halt.

(iii-c) If (18) holds, let $\bar{\lambda} := \hat{\lambda}$. Go to Step 5.

Step 5:

(i) If $J = \emptyset$, halt.

(ii) Else find $j^* \in J$ such that

$$p_{j^*} = \max\{p_j | j \in J\}$$

holds. Ties are broken by choosing the one with largest due date. Let $R := R - p_{j^*}$, $s := j^* s$ (i.e., j^* is appended to the beginning of s) and $J := J - \{j^*\}$. Return to Step 3. □

Theorem 1. Procedure SOLVEP correctly computes an optimal schedule of Problem P in $O(n^2 \log n)$ time. In addition, Procedure SOLVEP is strongly polynomial.

Proof: In order to prove the correctness of SOLVEP, we only have to show that $\lambda_{\min}$ is found in Step 4 (iii-b), unless (10) or (11) holds. Suppose otherwise. Then Procedure SOLVEP halts in Step 5 (i) with the interval $(\underline{\lambda}', \bar{\lambda}']$, where $\underline{\lambda}' < \bar{\lambda}'$ holds. Notice that SOLVEP has computed $\pi(\lambda_{\min})$ when it halts since SOLVEP follows the same computation path as SOLVEDEAD($\lambda_{\min}$) takes. Note that $\lambda_{\min}$ clearly satisfies

$\underline{\lambda}' < \lambda_{\min} \leq \bar{\lambda}'$. The fact that SOLVEP ends with an interval $(\underline{\lambda}', \bar{\lambda}']$ implies that SOLVEDEAD(λ) for all $\lambda \in (\underline{\lambda}', \bar{\lambda}']$ follows the same computation path. Thus, $\pi(\lambda_{\min})$ is optimal to $P(\lambda)$ for all $\lambda \in (\underline{\lambda}', \bar{\lambda}']$. $\pi(\tilde{\lambda})$ for $\tilde{\lambda}$ with $\underline{\lambda}' < \tilde{\lambda} < \lambda_{\min}$, however, does not satisfy $\alpha_1 T_{\max}(\pi'(\tilde{\lambda})) + \beta_1 \geq \alpha_2 F(\pi(\tilde{\lambda})) + \beta_2$ by definition of $\lambda_{\min}$ in (12). This is a contradiction.

We shall then analyze the running time. Since minimizing the maximum tardiness can be done by rearranging jobs in nondecreasing order of their due dates (see Jackson [4]), Step 0 requires $O(n \log n)$ time. Step 1 also requires $O(n \log n)$ time since Procedure SOLVEDEAD(λ) requires $O(n \log n)$ time by Lemma 1. Step 2 also requires $O(n \log n)$ time to arrange jobs in nondecreasing order of their due dates. Step 3 requires constant time. If we use a 2-3 tree to keep the set J (see the books by Aho, Hopcroft and Ullman [1] and Knuth [5], adding an element to J, deleting an element from J and taking the maximum of p_j over J are all done in $O(\log n)$ time. Therefore Step 4 (i) and (ii) can be done in $O(\log n)$ time. Step 4 (iii) requires $O(n \log n)$ time since SOLVEDEAD($\hat{\lambda}$) and SOLVEDEAD($\hat{\lambda} - 1$) require $O(n \log n)$ time. Hence, each application of Step 4 requires $O(n \log n)$ time. Step 5 requires $O(\log n)$ time. Since the loop of Steps 3, 4 and 5 is repeated $O(n)$ times, $O(n^2 \log n)$ time is required in total.

It is obvious that Procedure SOLVEP is strongly polynomial, since the running time depends only on the number of input data (i.e., $O(n)$), and the size of the numbers generated during the algorithm is clearly polynomially bounded. □

References

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

[2] J. Blazewicz, W. Cellary, R. Slowinski and J. Weglarz, Scheduling under resource constraints-deterministic models, *Annals of Operations Research*, 7(1986).

[3] D. Gusfield, Parametric combinatorial computing and a problem of program module distribution, *J. ACM*, 30 (1983), 551-563.

[4] J.R. Jackson, Scheduling a production line to minimize maximum lateness, Research Report 43, *Management Science Research Project*, Univ. of California, Los Angeles, 1955.

[5] D.E. Knuth, The Art of Computer Programming, Vol. 3: *Sorting and Searching*, Addison-Wesley, 1973.

[6] N. Megiddo, Combinatorial optimization with rational objective functions, *Math. Oper. Res.*, 4(1979), 414-424.

[7] N. Megiddo, Towards a genuinely polynomial algorithm for linear programming, *SIAM J. Computing*, 12 (1983), 347-353.

[8] H. Nakayama, On the components in interactive multiobjective programming methods (in M. Grauer, M. Thompson, A.P. Wierzbicki, editors: Plural Rationality and Interactive Decision Processes, Proceedings, 1984), Springer-Verlag, Berlin, 1985.

[9] R.T. Nelson, R.K. Sarin and R.L. Daniels, Scheduling with multiple performance measures: the one-machine case, *management Science*, 32 (1986), 464-479.

[10] S.S. Panwalkar, R.A. Dudek and M.L. Smith, Sequencing research and the industrial scheduling problem, in: S.E. Elmaghraby (ed.), Symposium on the Theory of Scheduling and its Applications, Springer-Verlag, Berlin, 1973.

[11] Y. Sawaragi, H. Nakayama and T. Tanino, Theory of Multiobjective Optimization, Academic Press, New York, 1985.

[12] W.E. Smith, Various optimizers for single-stage production, *Naval Res. Logist. Quart.*, 3 (1956), 59-66.

[13] R.E Steuer and E.V. Choo, An interactive weighted Chebyshev procedure for multiple objective programming, *Mathematical Programming*, 26 (1983), 326-344.

[14] E. Tardos, A strongly polynomial algorithm to solve combinatorial linear programs, *Operations Research*, 34 (1986), 250-256.

[15] L.N. Van Wassenhove and L.F. Gelders, Solving a bicriterion scheduling problem, *European J. Opl. Res.*, 4 (1980), 42-48.

[16] A.P. Wierzbicki, Penalty methods in solving optimization problems with vector performance criteria, Proceedings of the 6th IFAC World Congress, Cambridge-Boston, 1975.

[17] A.P. Wierzbicki, Basic properties of scalarizing functionals for multiobjective optimization, Mathematische Informationsforschung und Statistik. *Optimization*, 8 (1977), 55-60.

[18] A.P. Wierzbicki, On the use of penalty functions in multiobjective optimization, In W. Oettli, F. Steffens, et al., editors: Proceedings of the 3rd Symposium on Operational Research, Universität Mannheim, Athenaum, 1978.

[19] A.P. Wierzbicki, A mathematical basis for satisficing decision making, *Mathematical Modelling,* 3 (1982), 391-405.

[20] A.P. Wierzbicki, On the completeness and constructiveness of parametric characterizations to vector optimization problems, *OR Spektrum,* 8 (1986), 73-87.

[21] A.P. Wierzbicki and A. Lewandowski, Dynamic Interactive Decision Analysis and Support, Working Paper, IIASA, Laxenburg, Austria, 1987.